Imitation Learning with Keras

Carlos Matheus Barros da Silva, *Computer Engineering Bachelor Student of ITA*
Prof. Marcos Ricardo Omena de Albuquerque Máximo

***Abstract*—This paper evaluates two Keras' Neural Network by test it in different scenarios with two simple tests and an imitation learning problem.**

It was observed that the Neural Network worked fine for those purposes, and in some case, the result was really good, in the imitation learning case, for example.

***Index Terms*—Simple Evolution Strategy, SES, Covariance Matrix Adaptation Evolution Strategy, CMA-ES, optimization**

I. Introduction

Neural networks (NN) are computing systems vaguely inspired by the biological neural networks and astrocytes that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, an image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows the use of distributed training of deep-learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU).

II. Neural Network Implementation

The implementation was based on the file *imitatino learning*. The essence of the implementation is shown in Code 1.

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

# Adds the first layer
# The first argument refers to the number of neurons in
    this layer
# 'activation' configures the activation function
# input_shape represents the size of the input
# kernel_regularizer configures regularization for this
    layer
model.add(layers.Dense(75, activation=activations.linear,
    input_shape=(1,)))
model.add(layers.LeakyReLU(alpha))
model.add(layers.Dense(50, activation=activations.linear,
    input_shape=(75,)))
model.add(layers.LeakyReLU(alpha))
model.add(layers.Dense(20, activation=activations.linear,
    input_shape=(50,)))
model.compile(optimizer=optimizers.Adam(), loss=losses.
    mean_squared_error, metrics=[metrics.binary_accuracy])

history = model.fit(input, expected_output, batch_size=
    num_cases, epochs=num_epochs)

input_predict = np.matrix(np.arange(0, input[-1] + 0.001,
    0.001)).T
output = model.predict(input_predict)

# Comparing original and copied joint trajectories to
    evaluate the imitation learning
for joint in right_leg_joints:
    plt.figure()
    plt.plot(input, expected_output[:, joints_dict[joint]]
    * 180 / pi)
    plt.plot(input_predict, output[:, joints_dict[joint]] *
     180.0 / pi)
    plt.grid()
    plt.title(joint)
    plt.xlabel('Time (s)')
    plt.ylabel('Joint Position')
    plt.legend(['Original', 'Neural Network'])
    plt.savefig(os.path.join('imitation_learning_result',
     joint + '.' + fig_format), format=fig_format)
plt.show()
```

Code 1. Code of *imitatino learning*

III. Neural Network Analysis

A. Keras' Neural Network Analysis without regularization

In this analysis was used two test functions: *sum greater than* and *xor*. This initial analysis is not using regularization.

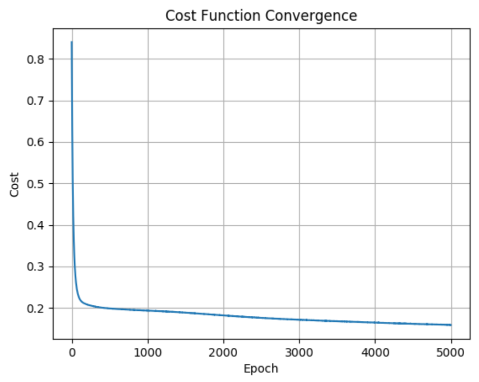

Fig. 1. Convergence of cost function on greater than function test case, when it is not using regularization.

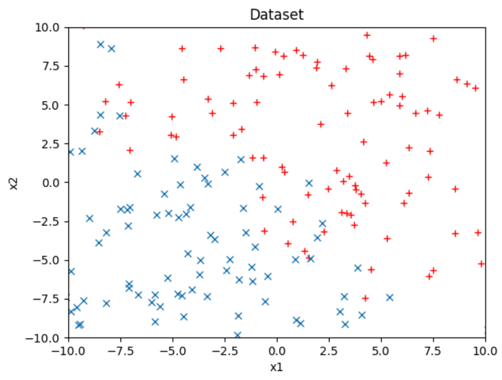

Fig. 2. Dataset of greater than function.

The Keras' Neural Network performed regular on *sum greater than* function. On the graphs represented by the images from Image 1 to Image 3. It is possible to verify that the result is overfitted on the Image 3 and the convergence is not so fast on the Image 1.

The Neural Network performed regular on *xor* function. On the graphs represented by the images from Image 4 to Image

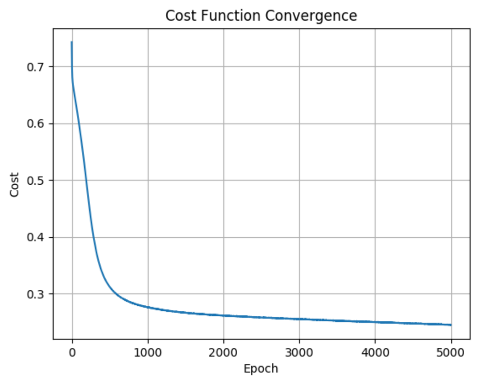

Fig. 4. Convergence of cost function on xor function test case, when it is not using regularization.

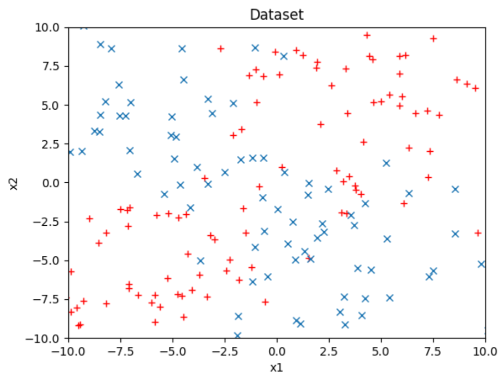

Fig. 5. Dataset of xor function.

6. It is also possible to see that in this case, it also heaped some overfit causing some distortion and leading to some mistakes on the data set on the graph represented by the Image 6.

B. *Keras' Neural Network Analysis without regularization*

In this analysis was used the same two test functions: *sum greater than* and *xor*. But now using regularization

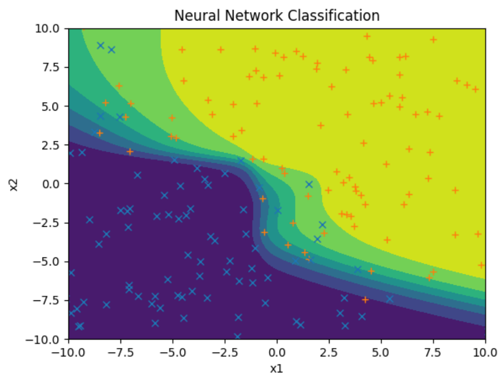

Fig. 3. Neural Network Classification on greater than function, when it is not using regularization.

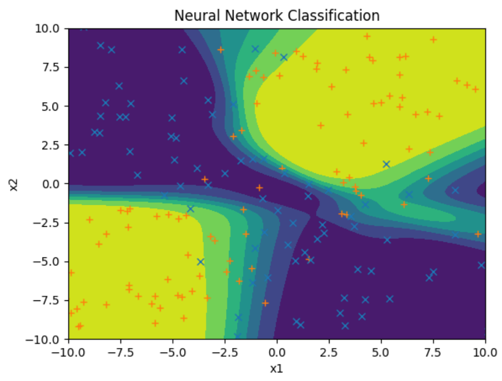

Fig. 6. Neural Network Classification on xor function, when it is not using regularization.

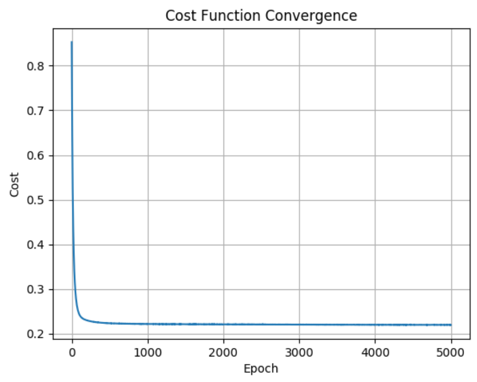

Fig. 7. Convergence of cost function on greater than function test case, when it is using regularization.

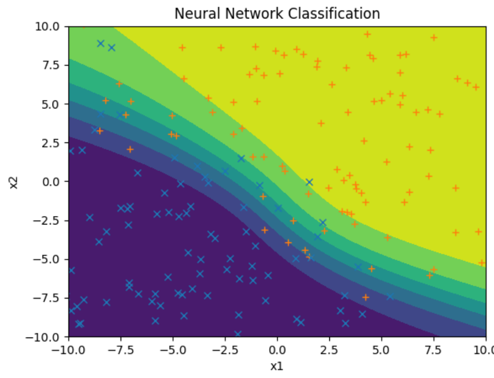

Fig. 8. Neural Network Classification on greater than function, when it is using regularization.

$\lambda_{l_2} = 0.002$.

The Keras' Neural Network performed well on *sum greater than* function. On the graphs represented by the images from Image 7 to Image 8. It is possible to verify that the result now is much less overfitted on the Image 8, it is much softer, and the convergence is now faster on the Image 7.

The Neural Network performed well on *xor* function. On the graphs represented by the images from Image 9 to Image 10. It is also possible to see that in this case, it also heaped much less overfit leading to a much softer image on the graph represented by the Image 10.

C. Keras' Neural Network Analysis in Imitation Learning

In order to do the Imitation Learning, it was used the Code 1 implementation.

It was made using Keras, not using regularization, and using mean squared error.

The result of this Neural Network on the robot movement can be seen on the graphs from Image 11 to Image 15.

IV. Conclusion

It was clear, therefore, that the Keras' Neural Network worked as expected. Both test cases (greater than function

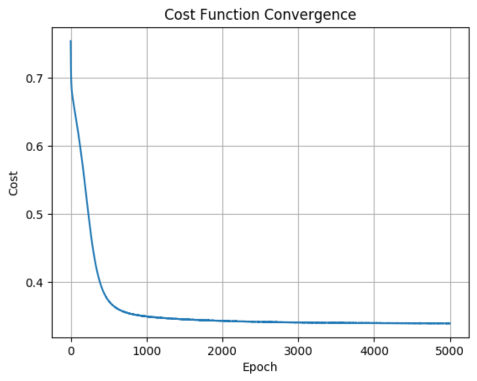

Fig. 9. Convergence of cost function on xor function test case, when it is using regularization.

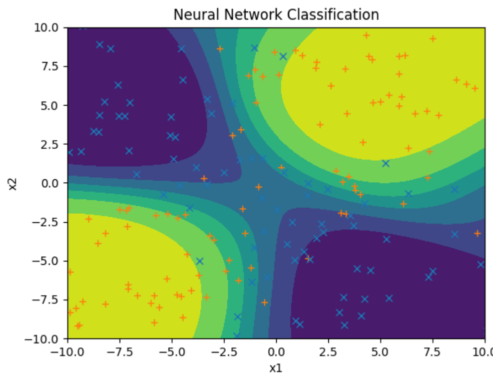

Fig. 10. Neural Network Classification on xor function, when it is using regularization.

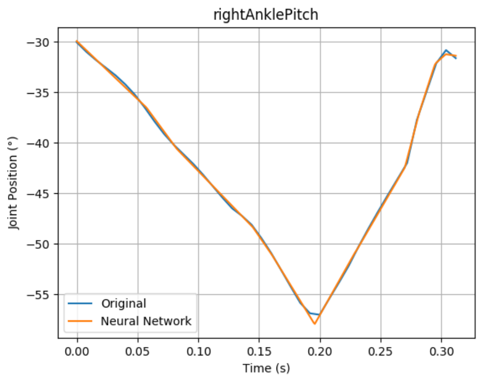

Fig. 11. Neural Network's movement imitation of robot's right ankle pitch

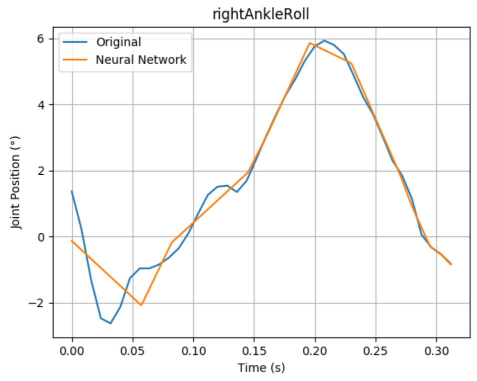

Fig. 12. Neural Network's movement imitation of robot's right ankle roll

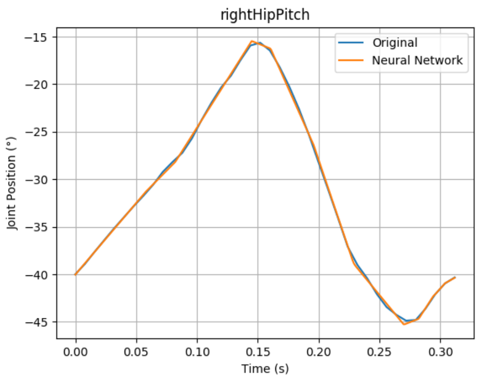

Fig. 13. Neural Network's movement imitation of robot's right Hip pitch

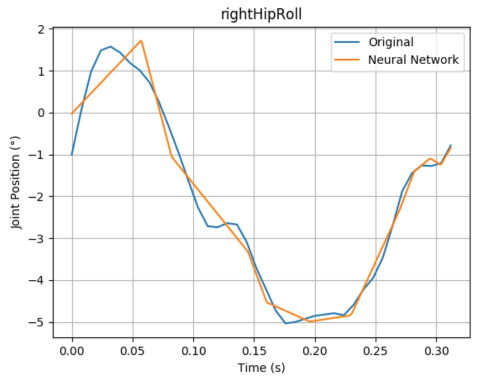

Fig. 14. Neural Network's movement imitation of robot's right hip roll

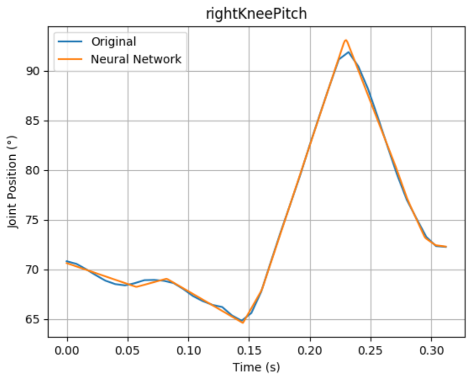

Fig. 15. Neural Network's movement imitation of robot's right Knee pitch

and xor function) the Neural Network worked as well, with a much better result with regularization, because without regularization the results were overfitted.

For the Imitation Learning with Keras, the results were good in some cases and very precise in most some cases.