Neural Networks

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

***Abstract*—This paper evaluates a shallow Neural Network by test it in different scenarios with two simple tests and a color segmentation test.**

It was observed that the Neural Network worked fine for those purposes, and in some case, the result was really good, in the greater than function test case, for example.

For the color segmentation, the Neural Network also provided a concise well-segmented result.

***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.

An NN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common NN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called ’edges’. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the NN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board, and video games and medical diagnosis.

II. Neural Network Implementation

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

The Keras' Neural Network performed regular on *sum greater than* function. On the graphs represented by the images

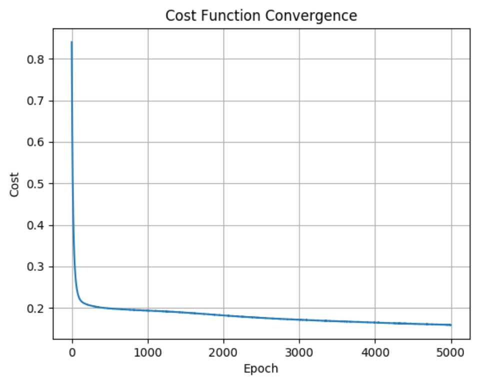

Fig. 1. Convergence of cost function on greater than function test case, when it is not using regularization.

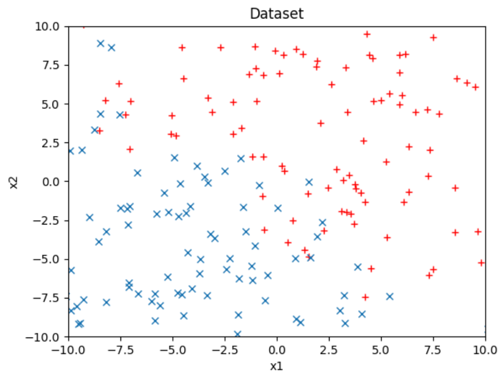

Fig. 2. Dataset of greater than function.

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 6. It is also possible to see that in this case, it also heaped some overfit causing some distortion and leading to some mistakes

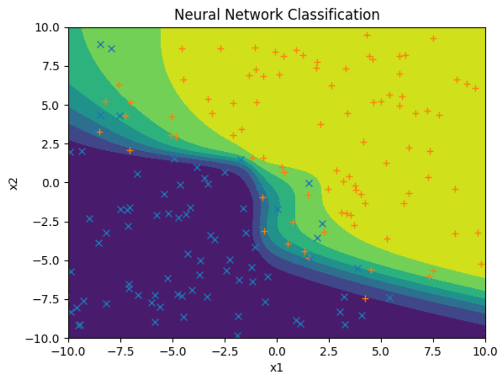

Fig. 3. Neural Network Classification on greater than function, when it is not using regularization.

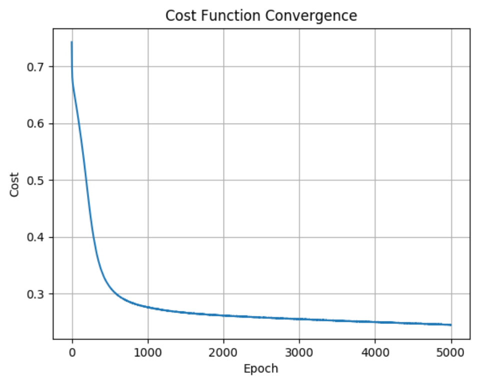

Fig. 4. Convergence of cost function on xor function test case, when it is not using regularization.

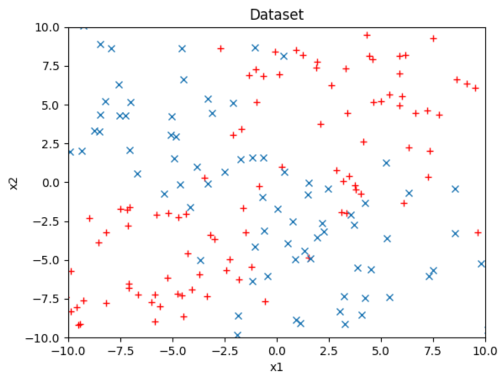

Fig. 5. Dataset of xor function.

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 $\lambda_{l_2} = 0.002$.

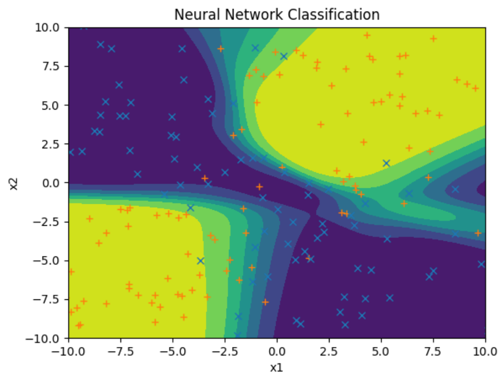

Fig. 6. Neural Network Classification on xor function, when it is not using regularization.

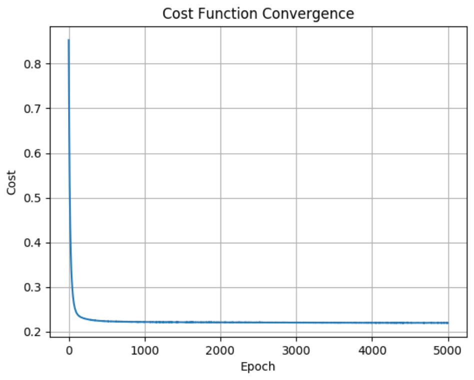

Fig. 7. Convergence of cost function on greater than function test case, when it is using regularization.

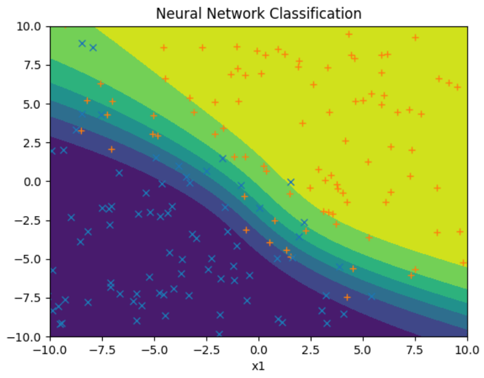

Fig. 8. Neural Network Classification on greater than function, when it is using regularization.

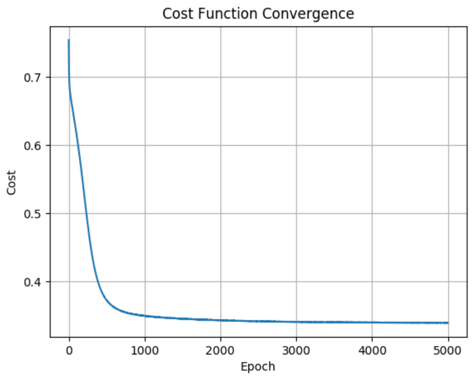

Fig. 9. Convergence of cost function on xor function test case, when it is using regularization.

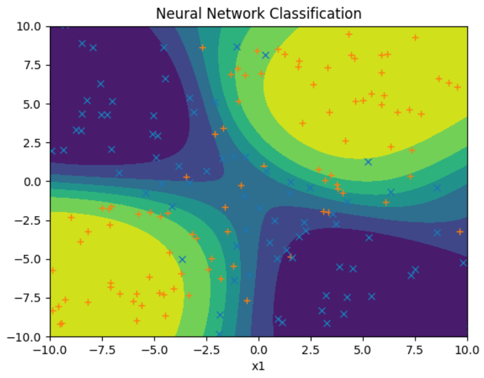

Fig. 10. Neural Network Classification on xor function, when it is using regularization.

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 moviment can be seen on the graphs from Image **??** to Image **??**.

IV. Conclusion

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

For the color segmentation, the Neural Network also provided a concise well-segmented result.