ACT 3 - Anomalous Detection LSTM

```
Eric Gómez - A01378838
Felipe Osornio - A01377154
Rafael Moreno - A01378916
Uriel Pineda - A01379633
Hector Hernandez - A01374009
```

Introduction

To this task, a comparison was made between two anomalous detection system using a dataset of vibration sensor readings used by the NASA Acoustics and Vibration Database. The analysis was made using the ARIMA and the Long Short-Term Memory. This latter is an artificial recurrent neural network arquitecture that is well-suited to classifying, processing and making pediction based on time series data.

In addition, the execution time, CPU consumption and the accuracy that existed between each of them were measured. This comparison is important because the ARIMA focuses mainly on obtaining results using some statistical techniques; such as variation and regression. While in the case of the LSTM, its main function is to execute epochs. Epochs are iterations through the entire neural network that are dedicated to defining the entire batch training data set. It is also important to mention that the size of the batch depends on how many item / values there are within it.

Methodology

Of this first part we implemented the ARIMA and the LSTM to make the anomalous detection inside the vibration sensor dataset. The most important improvement in this part was the modularity refactoring between the algorithms calls (ARIMA, LSTM) and the parameters (ARIMA, LSTM).

After that, we compare the execution time and CPU consumption of these two anomalous detection systems during the training and the predition.

Execution time

```python
def exec_time(func, args:list) -> str:
    '''
    Function that measures execution time of a given function
    '''
    intiTime = time()
    try:
        func(*args)
    except Exception as e:
        print('ERROR: ',e)
    elapsedTime = round(time()-intiTime, 2)
    return str(elapsedTime)+'s'
```

CPU consumption

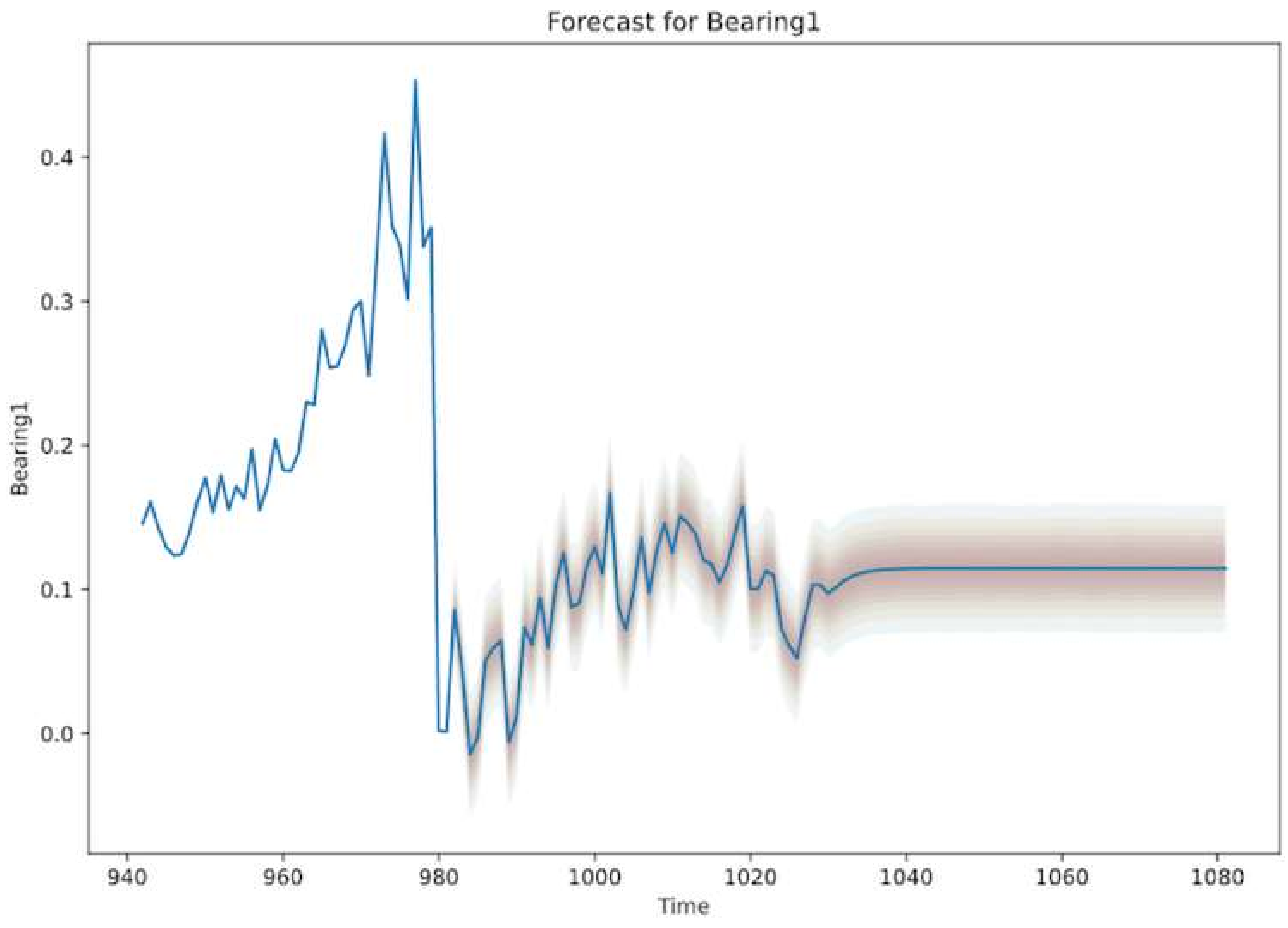
```python
def get_cpu_utilization() -> Thread:

    def cpu_util():
        cpu_data = []
        while(not IS_Process_end):
            sleep(1)
            cpu_data.append(cpu_percent())
        print('CPU utilization: ', round(mean(cpu_data),2),'%')

    return Thread(
        target=cpu_util
    )
```

In addition to measuring performance in terms of runtime and CPU consumption, the accuracy between the two models was also compared and measured using the minimum loss error function. In the case of ARIMA, as it's a statistical and iterative model; an optimal predicted standard deviation was obtained. Even as can be seen in the following graph the bearing was adjusted until a stable pattern was found.

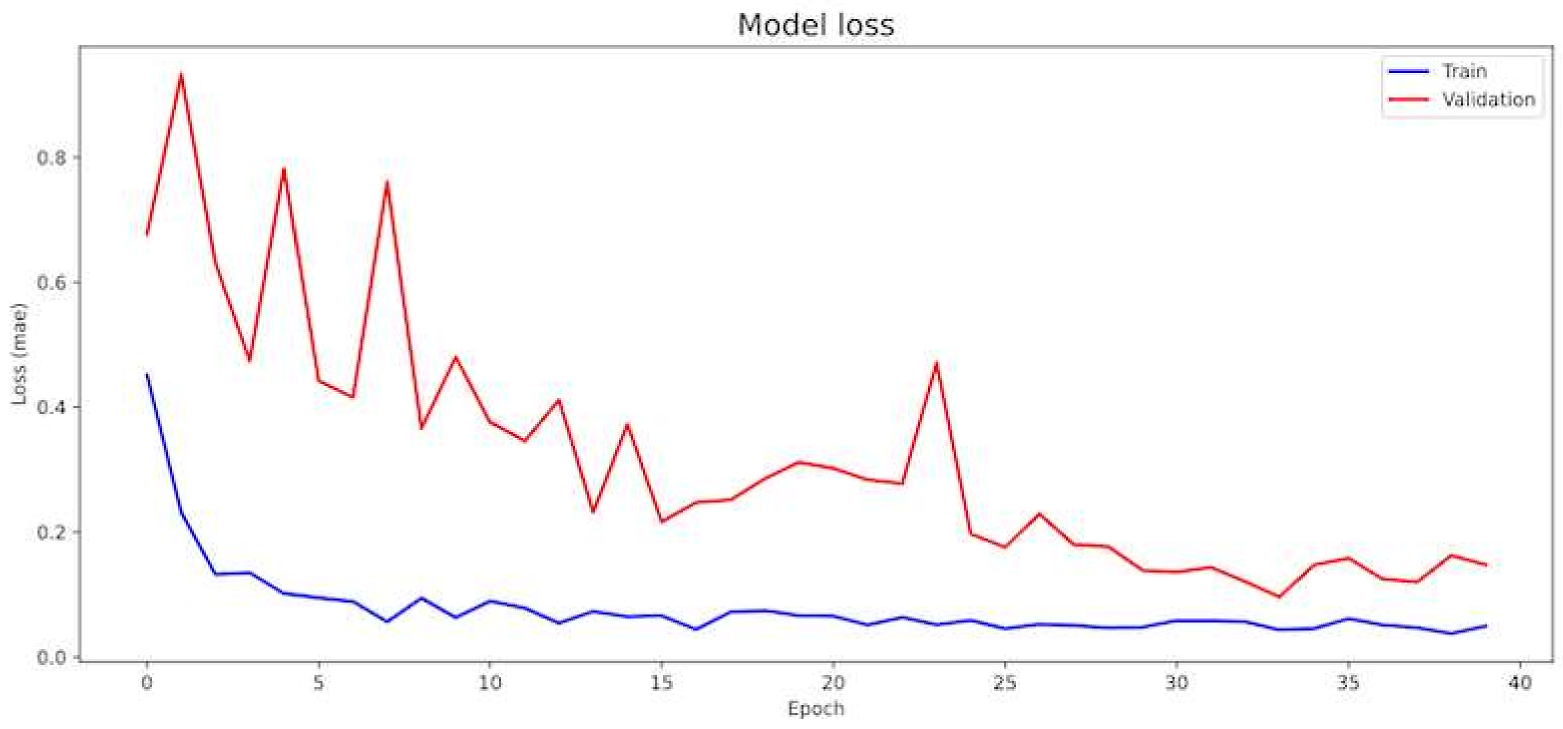
Figure 1



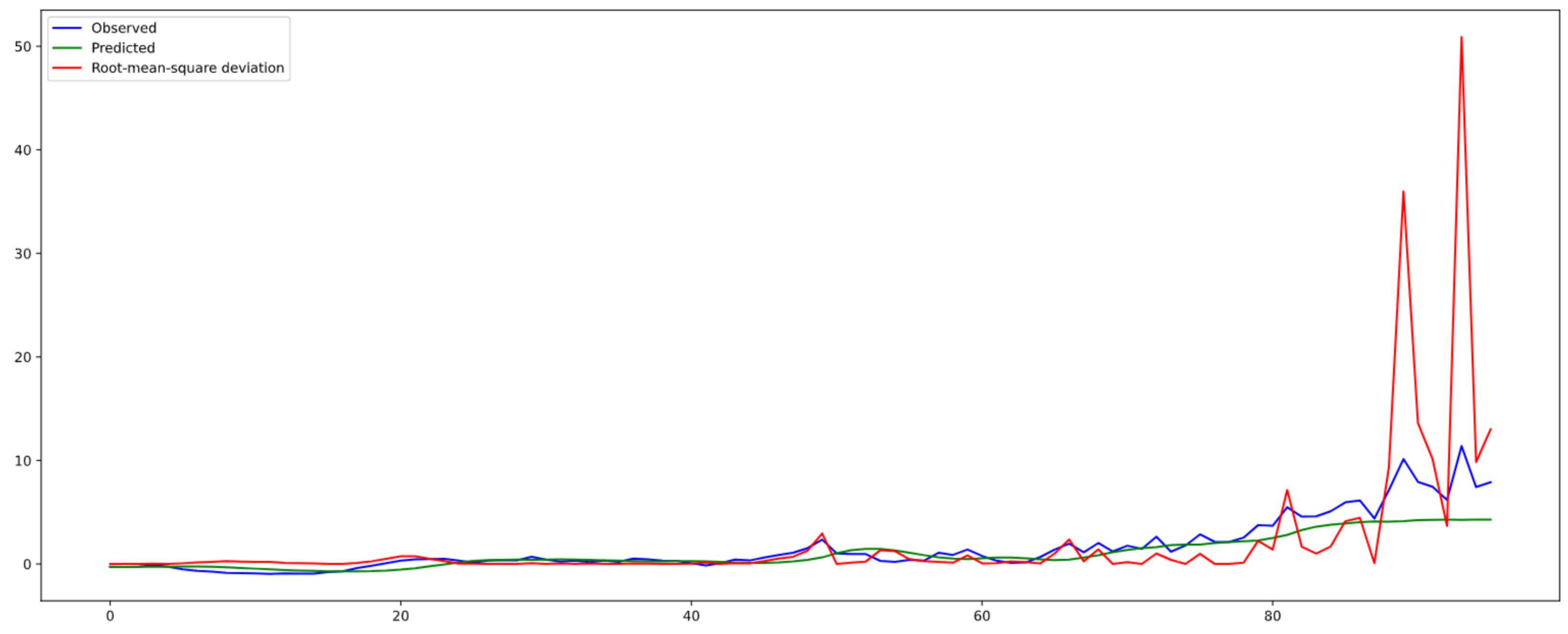
In the case of LSTM, in order to optimize its accuracy, the information was normalized to avoid a considerable difference between one value and another. Likewise, 70% of the dataset was taken for training with a batch size of 80, leaving the training in 40 epochs and a number of 9 iterations for each epoch.

Finally, to the case of the LSTM the following two graphs were obtained:

Figure 2



In this graph he compares the error against the number of epochs executed between the training dataset and the test dataset.



Regarding this graph, the comparison is made between the observed, the predicted and the root-mean-square deviation.

Finds

Among the most important findings is the comparison of CPU consumption, the predicted standard deviation of the set and its execution time. This comparison is shown below.

```
CPU utilization:  21.72 %
All dataset average:  0.081
Predicted dataset average:  0.104
All dataset st_deviation:  0.0402
Predicted dataset st_deviation:  0.0308
ARIMA TIME:  87.1 s
```

```
CPU utilization:  60.42 %
All dataset average:  0.081
Predicted dataset average:  0.9853
All dataset st_deviation:  0.0402
Predicted dataset st_deviation:  1.4887
LSTM TIME: 60.54 s
```

To estimate a convenient batch size for the execution of the model, it is recommended that this value is between twice the epoch number as the maximum value and the epoch number as the minimum value. This can be represented as follows:

```
epoch * 2 >= batch_size >= epoch
```

Regarding the comparison between these two outputs are the following points:

- As the batch size increases, there will be a greater amount of noise, but less processing time for each item / value.
- Otherwise, if the batch size decreases, the noise number will be less, however the processing time increases.
- The LSTM has a higher CPU consumption, however the execution time decreases.
- On the other hand, the ARIMA has a lower CPU consumption, but the execution time increases.
- As can be seen in the following graph. As the number of epochs increases, the minimum loss erros function decreases, which indicates that the precision increases.

Finally, it can be seen that the LSTM has a prediction that yields more dispersed data and on average has higher values than the original dataset. As for the ARIMA, being a classical inference PML, the prediction of the data is less dispersed and the average of the values is more similar to the original dataset.

Conclusion

In conclusion, the ARIMA tends to generate estimated values that are less dispersed and a little closer to the original data set, because it is a classical inference method, that is, the work is more mechanical and it is widely possible that these data will be adjusted since As the iterations pass, the variation between these data decreases.

The LSTM is quite functional to make these kinds of predictions quickly in order to get a general idea. However, if you want to improve the accuracy you need to consider adding more epochs. Not taking into account that it consumes more CPU, in contrast to the ARIMA.