UNIVERSITÀ DI PISA

Computer Engineering

Intelligent Systems

# ***Project Report***

---

*Student*:
Ayoub El Ourrak

Academic Year: 2021/2022

# Contents

# 1 — Introduction

The objective of this project is to create and implement a series of intelligent systems capable of analyzing collected physiological signals' timeseries data to identify various characteristics. We have successfully addressed all the requirements outlined in the project specifications. Specifically, we accomplished the following tasks:

- 3.1 Section:
    1. Designing and developing two artificial neural networks known as **multi-layer perceptrons (MLPs)**. The first MLP estimates the **mean**, while the second MLP estimates the **standard deviation** of a person's (XX) electrocardiogram (ECG) during three specific activities (referred to as YYYY). These estimations are based on the sensor data stored in the sXX_YYYY_timeseries file. To train the system and enable accurate predictions, we employed **feature extraction techniques** and utilized **sequential feature selection (sequentialfs)** to identify the most significant features.
    2. Additionally, we designed and trained **two radial basis function (RBF) networks** with similar objectives as the MLPs mentioned above.
- 3.2 Section: in this section, our focus shifted to the design and development of a **multi-layer perceptron** that classifies a person's activity into one of three categories: 'sit,' 'walk,' or 'run.'
- 3.3 Section: here, we developed a **fuzzy inference system** capable of classifying a person's activity using the k most relevant features (where k is less than or equal to 5) from the feature set used to train the classifier mentioned in section 3.2.
- 4.1 Section: the goal of this section mirrors that of section 3.1, with the exception of using a different type of neural network known as a **convolutional neural network (CNN)** instead of an MLP. We specifically focused on estimating the value (either mean or standard deviation) that exhibited the poorest performance when using MLPs in section 3.
- 4.2 Section: we proceeded to design and develop a **recurrent neural network (RNN)** capable of predicting a person's ECG value at a

given time step (t) based on either a partial or complete set of signals present in the dataset. The RNN takes these signals at various time steps (t - k, ..., t) along with the corresponding ECG values as input and returns the ECG value at the subsequent time step (t + 1).

# 2 — Dataset

This chapter provides an overview of the data processing techniques employed to enable the predictions discussed in the following chapters. The data utilized in this project is stored in CSV format files. Each subject contributes three recordings, each of which is divided into two separate CSV files. One file contains the physiological signals (referred to as timeseries files), while the other file contains the corresponding target values (target files). Each recording is associated with a specific physical activity. Consequently, we have a total of 66 timeseries CSV files and 66 target CSV files, resulting in a combined total of 132 CSV files.

The following are the names and structures of the two CSV files for a single recording:

1. sXX_YYYY_timeseries: this file captures the physiological signals and is organized into columns. The first column denotes the timestamp, while the subsequent columns represent different physiological signals. These signals include:

    - pleth_1: Red wavelength Photoplethysmography (PPG) from the distal phalanx (first segment) of the left index finger palmar side, sampled at 500 Hz.
    - pleth_2: Infrared wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side, sampled at 500 Hz.
    - pleth_3: Green wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side, sampled at 500 Hz.
    - pleth_4: Red wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side, sampled at 500 Hz.
    - pleth_5: Infrared wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side, sampled at 500 Hz.
    - pleth_6: Green wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side, sampled at 500 Hz.
    - lc_1: Load cell proximal phalanx (first segment) PPG sensor attachment pressure, sampled at 80 Hz.
    - lc_2: Load cell (base segment) PPG sensor attachment pressure, sampled at 80 Hz.
    - temp_1: Distal phalanx (first segment) PPG sensor temperature in degrees Celsius, sampled at 10 Hz.

    - temp_2: Proximal phalanx (base segment) PPG sensor temperature in degrees Celsius, sampled at 10 Hz.
    - temp_3: Ambient temperature in degrees Celsius, sampled at 500 Hz.

2. sXX_YYYY_targets: This file contains the target time series. Similar to the timeseries file, the first column represents the timestamp, while the subsequent column contains the target time series:

    - ecg: Three-channel electrocardiogram (ECG) sampled at 500 Hz.

The dataset creation process involved two main steps:

1. Feature Matrix Creation: This step involved creating a feature matrix using the data from the timeseries files.

2. Feature Selection: The final step focused on selecting the most relevant features from the dataset, ensuring optimal performance in subsequent prediction tasks.

By systematically following these two steps, we successfully processed the data, enabling us to proceed with accurate predictions in the subsequent chapters.

## 2.1 Feature Matrix Creation

Features were extracted from the time-series of the eleven sensors divided into windows. In particular, we divided each time-series into 23 non-overlapping windows and then use other 22 non-overlapping windows of the same size but shifted half their width with respect to the previous ones. From each window we extracted the features in table 1 for each signal through the feature_extraction script.

To ensure the efficiency and accuracy of our feature set, we conducted an analysis of feature correlation. In cases where we found that one or more features exhibited a correlation value greater than 0.90, we opted to eliminate redundancy by retaining only one of the correlated features and discarding the rest.

Once the feature matrix was prepared, we proceeded with a matrix normalization process. This operation transformed the data by redistributing values onto a standardized scale. By achieving a balanced distribution, the training algorithm could generate more precise results, preventing any significant variations from negatively impacting the remaining data. Various

| Time Domain | Frequency domain |
|---|---|
| Mean | Mean |
| Standard deviation | Standard deviation |
| Maximum | Maximum |
| Minimum | Minimum |
| Median | Median |
| Geometric mean | Geometric mean |
| Harmonic mean | Harmonic mean |
| Trimmed Mean | Trimmed Mean |
| Skewness | Skewness |
| Kurtosis | Kurtosis |
| Interquantile range | Interquantile range |
|  | Occupied bandwidth |

**Table 1:** Features extracted

normalization algorithms such as z-score, scale, and range were applied to normalize the sample data for each feature.

The extracted features were organized into a matrix of dimensions (22(participants)*3(activities)*45(windows) x 84(total features)). The rows of the matrix represented tuples, while the columns corresponded to the individual features.

For the target values, specifically the mean and standard deviation of the ECG signal, we extracted them and arranged them as column vectors, as shown in the target_extraction script. The same script was used to create the target class, a column vector containing the corresponding activity types (1=run, 2=sit, 3=walk) for each data sample.

## 2.2 Feature Selection

Once the training dataset was ready, we launched sequentialfs to find the most relevant features for our purposes. To this extent, we used the features_selection and f_extract scripts setting to 10 the max number of features to look for. The procedure was repeated five times, then the most selected features were chosen as the winning ones.

# 3 — Estimating ECG using Neural Networks

In this section, we will dive into the design and implementation of the models necessary to fulfill the requirements outlined in section 3.1 of the specification. These models encompass:

1. Creation of two Multi-Layer Perceptron neural networks: Our objective is to develop two neural networks capable of estimating the mean value and standard deviation of the ECG using the provided dataset as input.
2. Creation of two Radial Basis Functions (RBFs) networks: Similarly, we aim to construct two Radial Basis Functions (RBFs) networks that can estimate the mean value and standard deviation, mirroring the aforementioned neural networks' capabilities.

## 3.1 Multi-Layer Perceptron neural networks

In order to find the best architecture for the neural network, we decided to use the **feedforwardnet** to predict both the mean value and the standard deviation. To test the model, we split the dataset into:

1. 80% for training
2. 10% for validation
3. 10% for testing

To get the best possible performance from the network, we tried tuning by changing the training algorithm and the number of neurons. The optimal parameters for the two networks are:

1. Mean:
    - **trainbr** as training algorithm.
    - **12** as number of neurons in a single layer.
2. Std:
    - **trainlm** as training algorithm.
    - [**27, 37**] as number of neurons in two layers.

And the results are showed in figure 1 and 2 respectively for mean value and standard deviation:

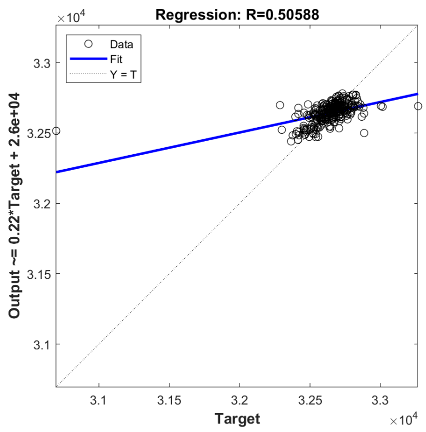


**Figure 1:** Mean regression plot

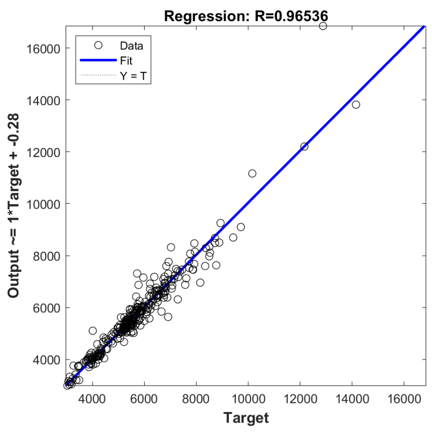


**Figure 2:** Std regression plot

## 3.2 Radial Basis Functions (RBFs) networks

The last step that concerns the 3.1 section of the specifics is the design of two Radial Basis Function networks, which are expected to estimate the mean and the standard deviation of the ECG of a person, just like the two MLPs in 3.1 section. We tried two different approaches, during the design of the two RBFNs:

1. RBFN (newrb) progressively constructs a Radial Basis Function Network by iteratively incorporating radbas neurons until either the mean squared error reaches a specified threshold or the maximum number of neurons is reached.

2. RBFN (newrb) trained with Bayesian Regularization (trainbr).

For both networks, we decided to set the max number of radbas neurons to 300 and tuned the spread value, by trying different value between the minimum and the maximum distance between points in the input space. The optimal parameters for the two networks are:

1. Mean:

   - **RBFN (newrb) without trainbr.**
   - **0.15** as spread value.

2. Std:

   - **RBFN (newrb) without trainbr.**

- **0.15** as spread value.

And the results are showed in figure 3 and 4 respectively for mean value and standard deviation:

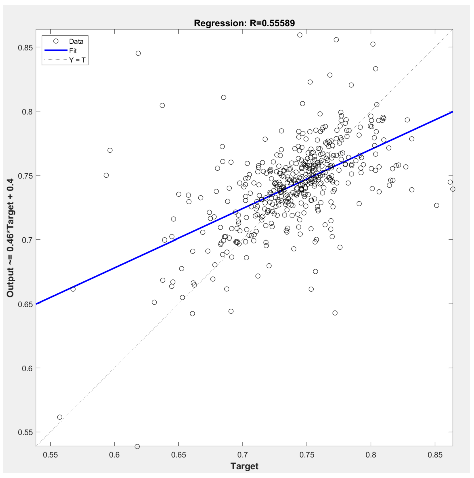


**Figure 3:** Mean regression plot

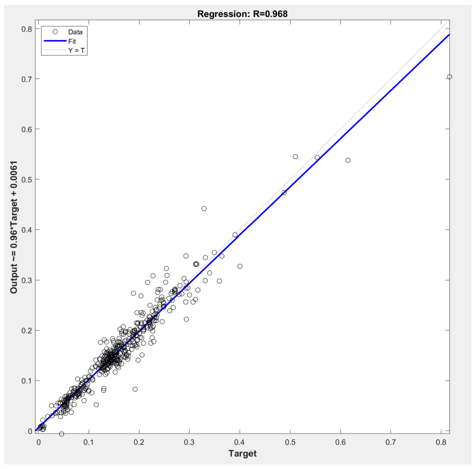


**Figure 4:** Std regression plot

# 4 — Determining a person's activity using neural networks

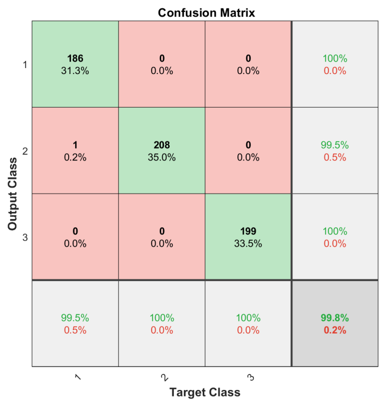
In this section, our focus is on **designing and developing a multi-layer perceptron (MLP)** capable of accurately classifying the activity performed by an individual during the recorded sessions. To achieve this, we employed the patternnet function to generate a pattern recognition network that predicts the subject's activity. For input to the network, we selected a feature matrix composed of 9 features, which were chosen during the feature selection process in Section 2. Regarding the target variable, we created a vector containing class labels representing the different activities performed by the subjects. To test the model, we split the dataset into:

1. 70% for training
2. 10% for validation
3. 20% for testing

To get the best possible performance from the network, we tried tuning by changing the training algorithm and the number of neurons. The optimal parameters are:

1. **traincgb** as training algorithm.
2. [**28, 20**] as number of neurons in two layers.

And the results are showed in figure 5:



Confusion Matrix

| Output Class \ Target Class | 1 | 2 | 3 | |
|---|---|---|---|---|
| 1 | 186<br>31.3% | 0<br>0.0% | 0<br>0.0% | 100%<br>0.0% |
| 2 | 1<br>0.2% | 208<br>35.0% | 0<br>0.0% | 99.5%<br>0.5% |
| 3 | 0<br>0.0% | 0<br>0.0% | 199<br>33.5% | 100%<br>0.0% |
| | 99.5%<br>0.5% | 100%<br>0.0% | 100%<br>0.0% | **99.8%**<br>**0.2%** |

**Figure 5:** Confusion Matrix

# 5 — Fuzzy Inference System

The objective of this section was to design and develop a fuzzy inference system capable of accurately classifying a person's activity into the categories of sit, walk, or run. To achieve this, we analyzed the distribution of the three most significant features within the feature set used to train the MLP classifier. These features were selected using the sequentialfs procedure on the MLP classification network.

To fulfill the goal, we constructed two distinct fuzzy inference systems:

1. A **Mamdani fuzzy system**, manually created using the Fuzzy Logic Designer tool.

2. A **TSK fuzzy system**, automatically generated using a grid-partitioning approach (genfis) and refined with an adaptive neuro-fuzzy algorithm (tunefis).

## 5.1 Mamdani Fuzzy System

We started by studying the distributions of the features for each activity through their histograms, as shown in the figure 6.

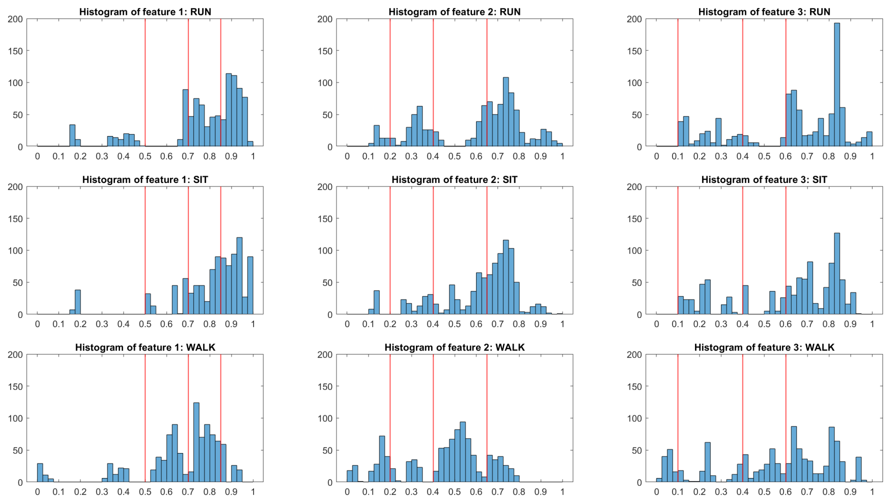


**Figure 6:** Distribution of features for each activity class

We partitioned the feature space into four zones and established **trapezoidal functions** as the membership functions. For the output membership functions, we opted for three **generalized bell** functions (sit, walk, run) because of their smoothness and symmetrical properties.

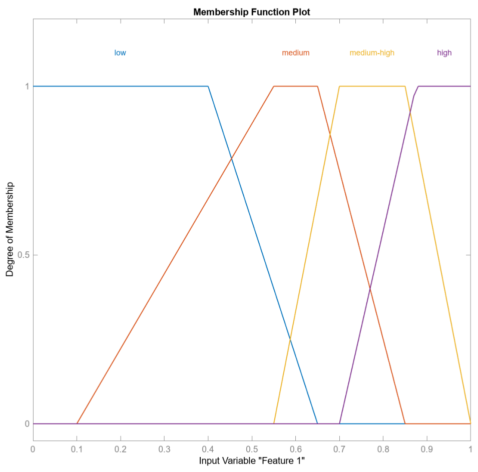


**Figure 7:** Membership functions of feature 1

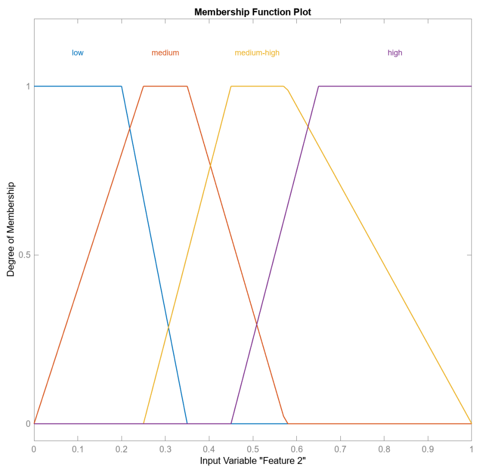


**Figure 8:** Membership functions of feature

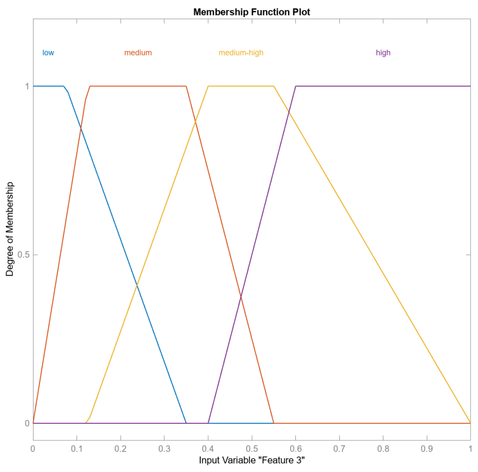


**Figure 9:** Membership functions of feature 3

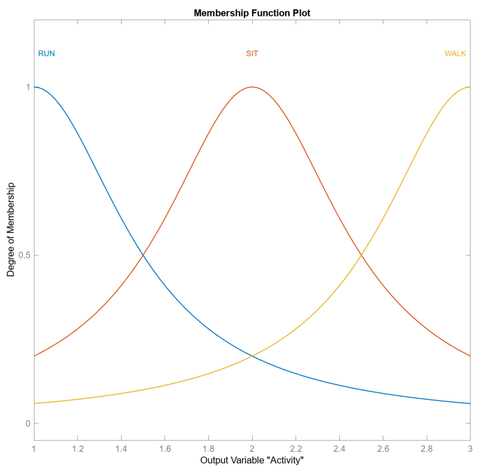


**Figure 10:** Membership function of the output variable

Due to the significant similarity in distributions, we conducted multiple experiments involving various rules with different weights and defuzzification methods. Despite our efforts, we were unable to attain an acceptable accuracy rate.

## 5.2 TSK Fuzzy System

In addition to manually designing a Mamdani fuzzy system using the Fuzzy Logic Designer toolbox, we explored an alternative approach. We automatically generated a FIS from the available data using the genfis function. In particular, we used the **Grid Partitioning method** creating 5 input membership functions.
Finally we tuned the system using an adaptive neuro-fuzzy algorithm through the tunefis function and obtained an overall accuracy of **92.49%**.

```
% Generate FIS
genfis_options = genfisOptions('GridPartition', ...
                               NumMembershipFunctions = 5);

fis = genfis(x, t_index, genfis_options);

% Tune FIS
[in, out, rules] = getTunableSettings(fis);
tunefis_options = tunefisOptions('Method','anfis');
tunefis_options.MethodOptions.EpochNumber = 30;
tunefis_options.UseParallel = true;
fis_final = tunefis(fis, [in; out], x, t_index,
    tunefis_options);
```

# 6 — Improve ecg estimation using convolutional neural networks

In this section, our focus shifted to estimating the mean of the ECG values using a **Convolutional Neural Network (CNN)** instead of the MLP utilized in section 3.1, which demonstrated poor performance in this task.

For the dataset, we segmented the biophysical signals of each time-series into 25 non-overlapping windows, subsequently grouping them together to serve as input for training the CNN.

To identify the optimal CNN architecture, we experimented with various network configurations. This involved modifying the network's layers, adjusting the number and size of filters, exploring different training algorithms, and testing various data normalization techniques. Our aim was to identify the configuration that yielded the best results.

Our Convolutional Neural Network (CNN) architecture comprises several key components:

- **1-D Convolution Layer:** This layer applies sliding convolutional filters to the 1-D input data.
- **Batch Normalization Layer:** We incorporated a batch normalization layer to accelerate the CNN's training process, reduce sensitivity to network initialization, and mitigate the issue of exploding gradients.
- **Leaky ReLU Layer:** which prevents the complete disappearance of gradient signals during training.
- **1-D Max Pooling Layer:** This layer facilitates downsampling by dividing the input into 1-D pooling regions and extracting the maximum value within each region.
- **1-D Global Average Pooling Layer:** This layer performs downsampling by computing the average of the time dimension of the input.
- **Dropout Layer:** Incorporated to combat overfitting.
- **Fully Connected Layer with Leaky ReLU Activation:** Serving as the hidden layer of the MLP, this fully connected layer employs the leaky ReLU activation function.
- **Fully Connected Layer with Regression Activation:** Representing the output layer of the MLP, this fully connected layer utilizes a regression activation function to generate the desired predictions.

The final architecture configuration is:

```
filter_size = 20;
num_filters = 50;
pool_width = 4;
layers = [
    sequenceInputLayer(11)

    convolution1dLayer(filter_size, num_filters, 'Stride', 2, 'Padding', 'same')
    batchNormalizationLayer
    leakyReluLayer
    maxPooling1dLayer(pool_width, 'Stride', pool_width, 'Padding', 'same')


    convolution1dLayer(filter_size, 2 * num_filters, 'Stride', 2, 'Padding', 'same')
    batchNormalizationLayer
    leakyReluLayer
    maxPooling1dLayer(pool_width, 'Stride', pool_width, 'Padding', 'same')


    convolution1dLayer(filter_size, 4 * num_filters, 'Stride', 2, 'Padding', 'same')
    batchNormalizationLayer
    leakyReluLayer
    maxPooling1dLayer(pool_width, 'Stride', pool_width, 'Padding', 'same')


    globalAveragePooling1dLayer
    fullyConnectedLayer(100)
    dropoutLayer(0.3)
    fullyConnectedLayer(1)
    regressionLayer
];
```

In our exploration of alternative network configurations, we opted to randomly experiment with different hyperparameter combinations. However, across all these variations, the performance did not align with that of the MLP, resulting in no improvement.

# 7 — Predict ecg value using recurrent neural networks

As outlined in the specification, the **recurrent neural network** is designed to accept input from one or more signals based on their values within specific time intervals [t - k, t]. Here, 't' represents the starting time of the interval (window), while 'k' denotes the window width.

In our pursuit of determining the optimal RNN architecture, we conducted experiments involving diverse network configurations. Among these configurations, we discovered the best result by utilizing the following configuration:

```
layers = [
    sequenceInputLayer(num_channels)
    lstmLayer(150)
    fullyConnectedLayer(1)
    regressionLayer
];
```

In the following figures we can observe the results obtained:

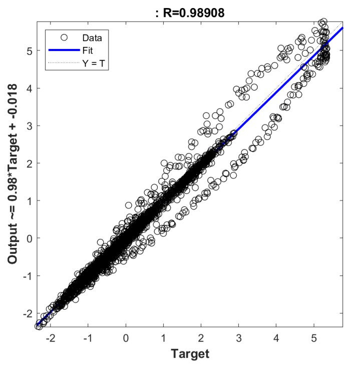


**Figure 11:** RNN regression plot

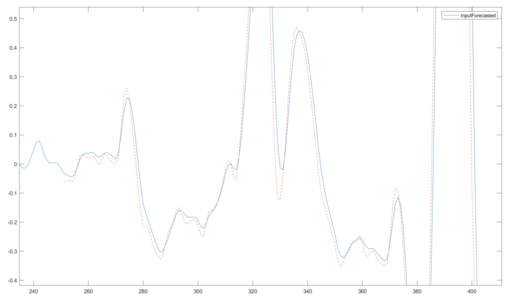


**Figure 12:** RNN one-step forecasting