Dynamic Modeling of Robot Arm using NARX Neural Network in MATLAB

Mehuli Lahiri

***Abstract*—This paper presents a dynamic modeling approach for a two-joint robot arm using a Nonlinear Autoregressive with Exogenous Input (NARX) Neural Network. The study utilizes MATLAB's `robotarm_dataset` to predict joint angles based on torque inputs. The results demonstrate high accuracy, achieving an R^2 score of 1.0 and very low error metrics.**

I. Introduction

Dynamic modeling of robotic systems is essential in control and automation. Neural networks, especially NARX models, offer significant advantages in capturing nonlinear and temporal dependencies. This work aims to model a robotic arm using a NARX architecture trained on real-time torque-angle data from MATLAB's built-in dataset.

II. Methodology

The methodology involves the use of a NARX neural network for time-series prediction. The system inputs are torque signals applied to the robotic joints, and the targets are the resulting joint angles.

A. Dataset

We used MATLAB's `robotarm_dataset`, which provides multivariate time-series data comprising joint torque inputs and angle outputs. The inputs are formatted as cell arrays suitable for sequential modeling.

B. Network Architecture

The NARX neural network includes:

- Input delays: 1 to 2 time steps
- Feedback delays: 1 to 2 time steps
- Hidden layer: 10 neurons

The network was trained using the Levenberg-Marquardt backpropagation algorithm.

C. Training Procedure

The input and target data were preprocessed using the `preparets` function. Data were split into 70% training, 15% validation, and 15% testing. Training performance was monitored and validated through MSE loss.

III. Implementation

The entire process was implemented in MATLAB using the Neural Network Toolbox. The model was trained, tested, and evaluated with various metrics including MSE, RMSE, MAE, and R^2. The code used for training and evaluation is shown in Fig. 1.

```
>> % Step 1: Load dataset
[X, T] = robotarm_dataset;

% Step 2: Convert to time series format
inputSeries = X;      % Torque inputs
targetSeries = T;     % Joint angles

% Step 3: Create NARX network
inputDelays = 1:2;     % Input delays
feedbackDelays = 1:2;  % Feedback delays
hiddenLayerSize = 10;
net = narxnet(inputDelays, feedbackDelays, hiddenLayerSize);

% Step 4: Prepare data for training
[inputs, inputStates, layerStates, targets] = preparets(net, inputSeries, {}, targetSeries);

% Step 5: Data division for training, validation, testing
net.divideParam.trainRatio = 0.7;
net.divideParam.valRatio   = 0.15;
net.divideParam.testRatio  = 0.15;

% Step 6: Train the network
[net, tr] = train(net, inputs, targets, inputStates, layerStates);

% Step 7: Test network
outputs = net(inputs, inputStates, layerStates);
performance = perform(net, targets, outputs);

% Step 8: Error metrics
targetVec = cell2mat(targets);
outputVec = cell2mat(outputs);
```

Fig. 1: MATLAB code for training the NARX network

IV. Results and Discussion

A. Evaluation Metrics

Table I summarizes the performance of the trained network.

TABLE I: Performance Evaluation Metrics

Metric	Value
Mean Squared Error (MSE)	1.0591e-08
Root Mean Squared Error (RMSE)	1.0291e-04
Mean Absolute Error (MAE)	6.2716e-05
R-squared (R^2)	1.0000

B. Visualization Results

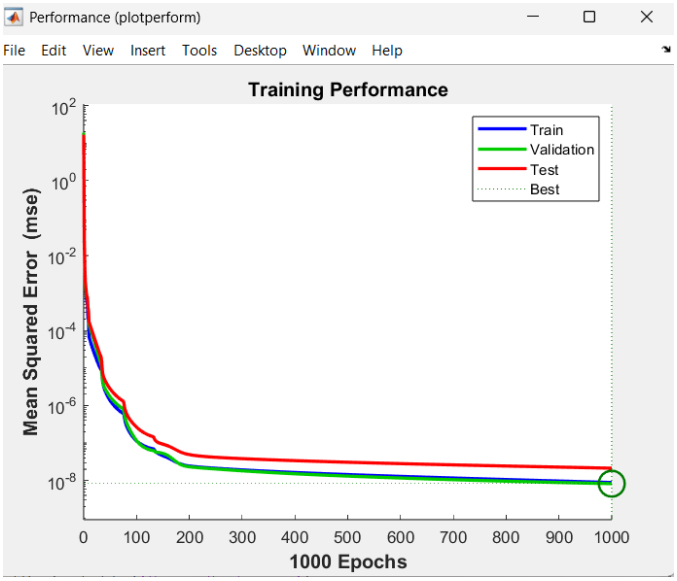

Fig. 2: Training performance (MSE vs epochs)

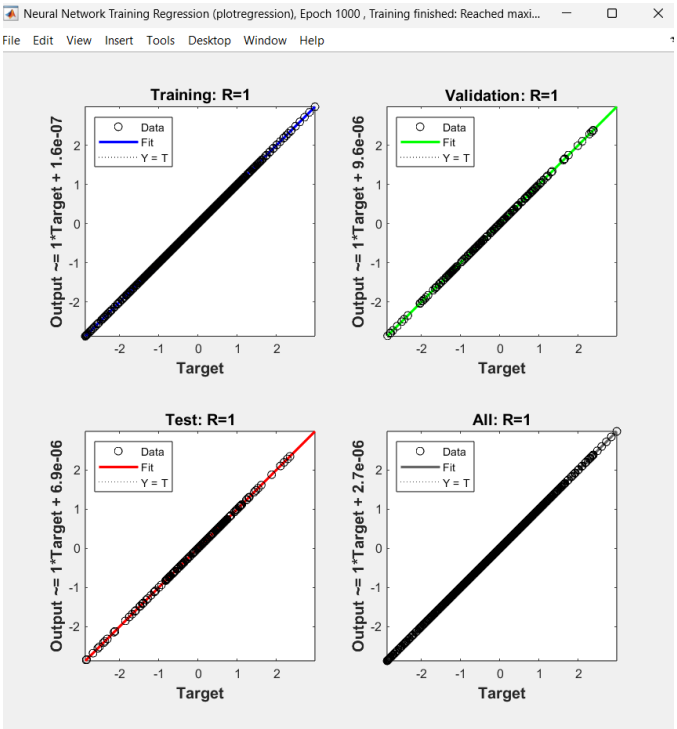

Fig. 3: Regression plot showing predicted vs. actual outputs

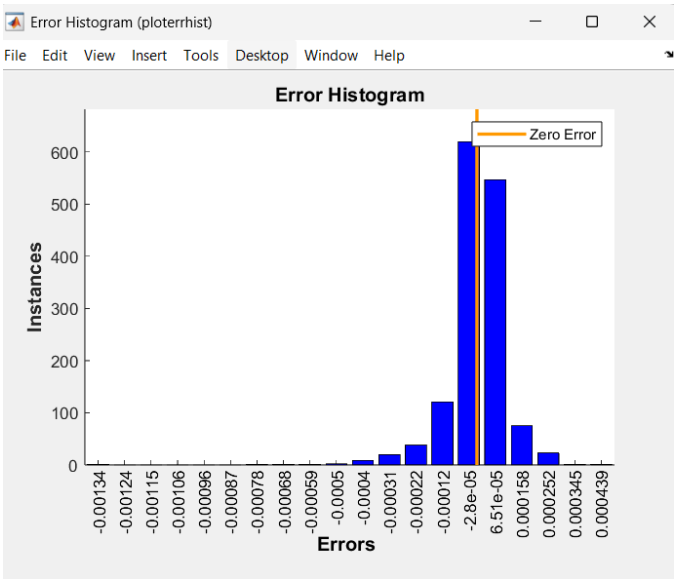

Fig. 4: Error histogram of prediction error

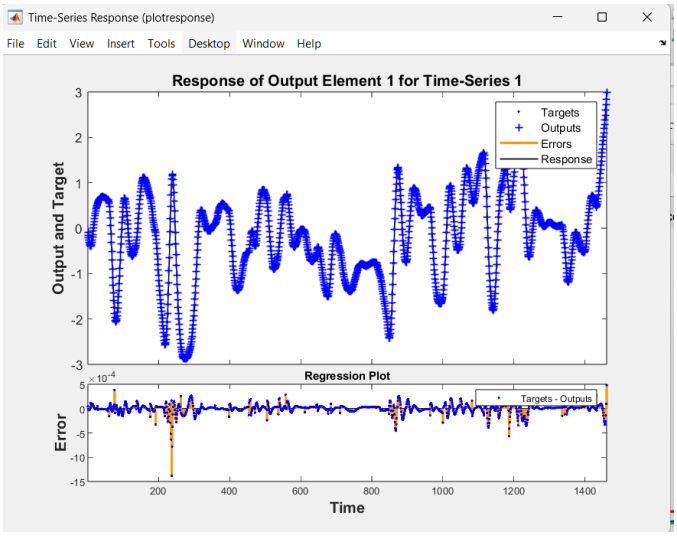

Fig. 5: Time-series plot of target vs predicted outputs

V. Conclusion

The NARX neural network proved highly effective for dynamic system modeling of the robot arm. With extremely low error rates and a perfect R^2 score, the model accurately captured the nonlinear dependencies between torque and joint angle. This confirms the viability of NARX models in robotic control applications.