MEWNet: Multivariate Ensemble Wavelet Network

saee.s.kamat

September 2024

1 MEWNet Documentation

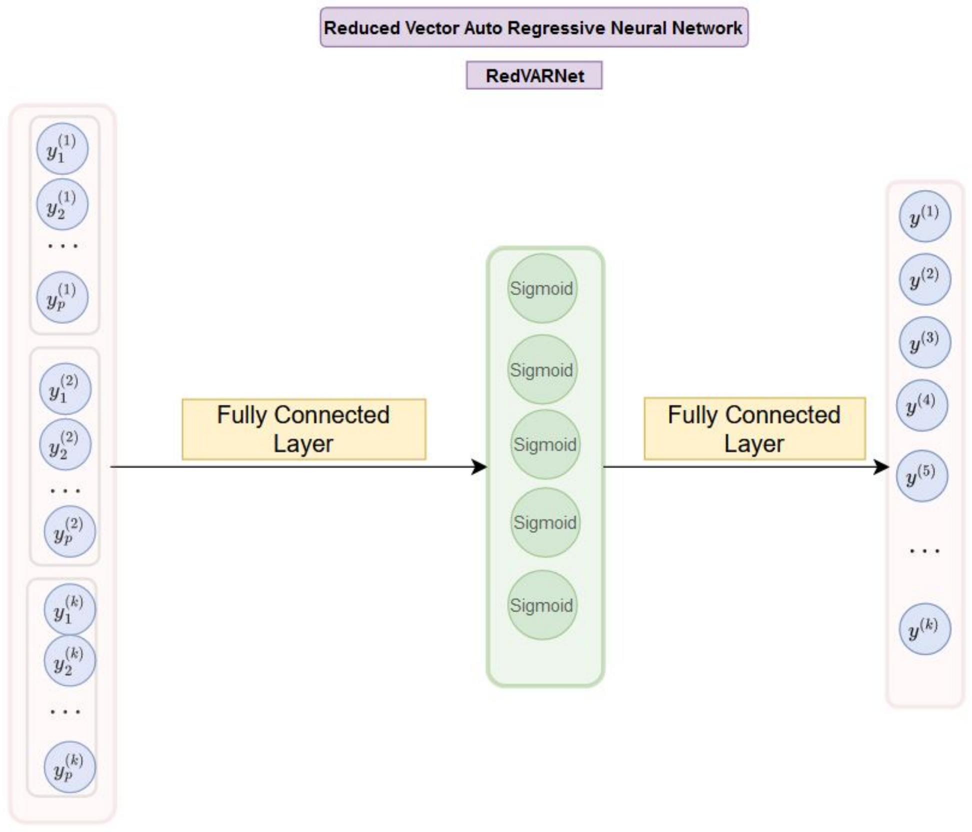
1.1 MyTimeSeriesDataset Class

The `MyTimeSeriesDataset` class is designed to handle time series data for training and validating neural network models. This class takes into account both endogenous and exogenous variables and provides methods for creating batches of data with specified lags and forecast horizons.

Class Methods:

- **preprocess mydata:** Makes train and test split on data according to user definied values. Makes transformations of differencing or standardising on each and returns transformed train and test data frames.
- **frame myseries:** Making time series suitable for supervised learning(Input and Target), using sliding window technique. Generates lagged input sequences for training the model.
- **getmyloaders:** Using data loaders to make batches of train and validation sets, while preserving the temporal relationship.

1.2 VARNeuralNetwork Class



The `VARNeuralNetwork` class is a neural network model tailored for inducing non linearity in classical vector autoregressive (VAR) model. It is designed for multistep forecasts for multivariate time series simultaneously, incorporating both endogenous and exogenous variables. Allows specification of input size, hidden size, and output size to adapt to different datasets and forecasting needs.

Notation:

- $y_i^{(l)}$: i^{th} lagged observation of l^{th} variable
- Sigmoid : Signoid activation function

Class Methods:

- **Constructor:** Accepts both endogenous and exogenous variables as input.

- **forward:** Utilizes specified lags of past observations of all variables in fully connected layer, and induces non linearity by using sigmoid activation function.

1.3 RedVARNet

Arguments:

- df = A pandas Data Frame with variables
- exotest = Test dataset for exogenous variable
- exo = list of exogenous variables(default None)
- batchsize = size of batches (default 1)
- standardise = Transform data by standardising (default False)
- differencing = Transform data by differenicng (default False)
- lags = No of previous models to consider(default 2)
- trainsize = Size of train set(default 0.8)
- valsize = Size of validation set(default 0.2)
- msteps = Length of horizon(default 4)
- hiddensize= Number of neurons in hidden layer ceil((lags * features + 1)/2)
- learningrate= Learning rate(default 0.001)
- numepochs= Number of epochs(default 100)
- optimizertype = Optimization technique : 'Adam' or 'SGD' default 'SGD'

Return values: Dataframe of forecasts

Algorithm 1 Algorithm for Reduced Vector Auto Regressive Neural Network

```
Require: Dataframe, exogenous test dataset, transformation flags, lags, horizon length
1: Rearrange the data frame by placing exogenous variables before endogenous variables.
2: Get training and validation batches from MyTimeSeriesDataset().
3: Instantiate the VARNeuralNetwork class with appropriate input size, hidden size, and output size.
4: Choose the optimization technique.
5: Train the model for the specified number of epochs.
6: Validate the model for the specified number of epochs.
7: Make a multistep forecast with exogenous variables for the specified horizon length.
8: Apply inverse transforms if any transformations (differencing/standardizing) are specified.
9: Return the forecast dataframe.
```

1.4 mywavetransform reduced

Arguments:

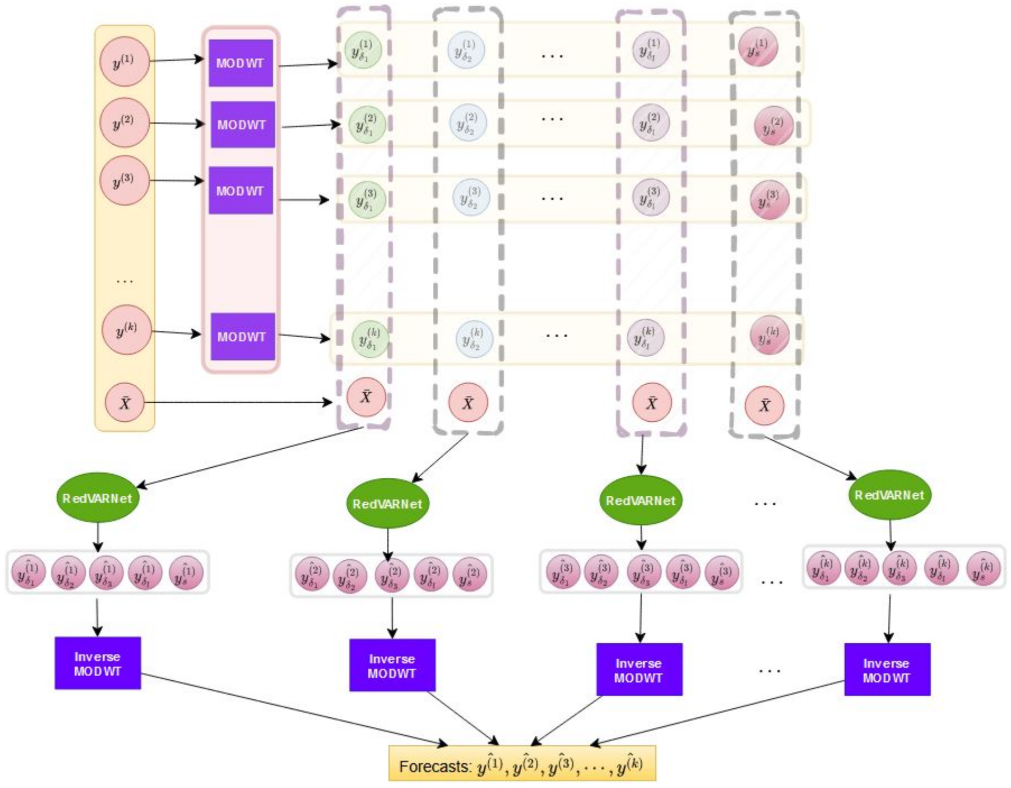
- df = A pandas Data Frame with variables
- exotest = Test dataset for exogenous variable
- wname = name of the wavelet used for decomposition(default 'Haar')
- lev = number of detailed decompositions required(default 4)
- exo = list of exogenous variables(default None)
- batchsize = size of batches (default 1)
- standardise = Transform data by standardising (default False)
- differencing = Transform data by differenicng (default False)
- lags = No of previous models to consider(default 2)
- trainsize = Size of train set(default 0.8)
- valsize = Size of validation set(default 0.2)
- msteps = Length of horizon(default 4)

- hiddensize= Number of neurons in hidden layer ceil((lags * features + 1)/2)
- learningrate= Learning rate(default 0.001)
- numepochs= Number of epochs(default 100)
- optimizertype = Optimization technique : 'Adam' or 'SGD' default 'SGD'

Return values: List of forecasts corresponding to each decomposition

Algorithm 2 Algorithm for using MOWDWT with RedVARNet

```
Require: Inputs
1: Subset the data frame excluding the exogenous variables
2: Apply MODWT transformation on each endogenous variable to decompose
   it ino lev + 1 many levels.
3: Make lev + 1 many new data frames which have same level decomposition
   of each variable
4: Append these data frames with exogenous variable and get multistep fore-
   casts for each multivariate time series from RedVARNet()
5: Apply inverse MODWT on ensemble of forcasted values of each decompo-
   sition of each variable and convert to a dataframe
6: Return the forecasted dataframe
```


MODWT
RedVARNet
Inverse MODWT
Forecasts: $\hat{y^{(1)}}, \hat{y^{(2)}}, \hat{y^{(3)}}, \cdots, \hat{y^{(k)}}$

Notation:

- l : level of decomposition
- k : number of endogenous variables
- $y^{(1)}, y^{(2)}, \cdots, y^{(k)}$: endogenous variables
- $\bar{X}$: collection of all exogenous variables
- `MODWT` : Maximal Overlap Discrete Wavelet Transform
- $y^{j}_{\delta_i}$: i^{th} detailed decompostion of j^{th} endogeous variable, where, $i = 1, 2, \cdots, l$ and $j = 1, 2, \cdots, k$
- y^{j}_{s} : Smooth decomposition of j^{th} endogeous variable, where, $j = 1, 2, \cdots, k$
- $y^{\hat{j}}_{\delta_i}$ forecasts of i^{th} detailed decompostion of j^{th} endogeous variable, where, $i = 1, 2, \cdots, l$ and $j = 1, 2, \cdots, k$

- $\hat{y_s^j}$: Forecast of Smooth decomposition of j^{th} endogeous
- `InverseMODWT` : Inverse transformation of MODWT

1.5 perform evaluation

Arguments:

- df = orginal dataset
- forecastdf = dataframe of forecasts
- horizonlength = horizon length

Return value: Forecasts evaluation metrics for each variable. The metrics include:

- MSE
- SMAPE
- MASE
- RMSE
- MAE
- SMDAPE
- Theilś U1
- CRPS

2 Project Report

Hybrid time series models effectively integrate traditional statistical techniques with advanced machine learning methods, making them particularly valuable for multivariate time series analysis. Traditional statistical models, such as Autoregressive Integrated Moving Average (ARIMA) and Vector Autoregression (VAR), excel in capturing linear relationships among variables and offer interpretability. However, they often struggle to address complex, non-linear patterns prevalent in many real-world datasets. Conversely, machine learning models, including neural networks, demonstrate superior performance in modeling these non-linear interactions and discovering intricate features. Yet, they typically require large datasets and may lack interpretability. By synergizing these approaches, hybrid models can encapsulate both linear and non-linear relationships, handle high-dimensional multivariate data, and enhance forecasting accuracy. Moreover, hybrid models enable regularization when data availability is limited and adapt dynamically to time-varying structures within the data.

A notable advantage of hybrid models is their capacity for robust uncertainty quantification and enhanced generalization. Traditional models provide clear interpretability and explicit uncertainty estimates, while machine learning methods add flexibility in feature extraction and adaptability to sudden shifts in the data. These capabilities are particularly advantageous in cross-domain applications, where hybrid models can strike a balance between rigid model assumptions and the flexibility needed for improved accuracy without overfitting. Thus, hybrid models effectively mitigate the biases associated with individual approaches, offering a more comprehensive solution for multivariate time series forecasting by leveraging the statistical rigor of traditional methods alongside the complex pattern recognition abilities of machine learning.

The **Vector Autoregressive (VAR)** model is a foundational tool in multivariate time series analysis, representing each variable as a linear function of its own past values and the past values of all other variables in the system. Mathematically, for a k-dimensional time series $Y_t = (y_{1t}, y_{2t}, \ldots, y_{kt})$, the VAR model of order p (VAR(p)) can be expressed as follows:

$$Y_t = c + A_1 Y_{t-1} + A_2 Y_{t-2} + \cdots + A_p Y_{t-p} + \epsilon_t,$$

where A_i are coefficient matrices, c is a vector of constants, and ϵ_t denotes the error term. To enhance the predictive capabilities of the VAR model, we propose integrating it with a **Multilayer Perceptron (MLP)**. This integration involves utilizing the residuals from the VAR model, denoted as ϵ_t, as features fed into the MLP, thereby enabling the MLP to approximate non-linear relationships that the VAR model alone may overlook. The MLP learns a mapping function defined as:

$$f(\epsilon_t) = W_2 \sigma(W_1 \epsilon_t + b_1) + b_2,$$

where W_1 and W_2 are weight matrices, b_1 and b_2 are bias vectors, and σ represents a non-linear activation function. This configuration helps capture complex interactions that would otherwise remain unmodeled.

When applying **wavelet filters** within the combined **VAR-NN (VAR Neural Network)** framework, the time series data undergoes decomposition into distinct frequency components via wavelet transforms. Let $W(Y_t)$ represent the wavelet-transformed series, where the operator W applies a wavelet filter to extract high-frequency and low-frequency components. These transformed components serve as inputs for both the VAR and neural network

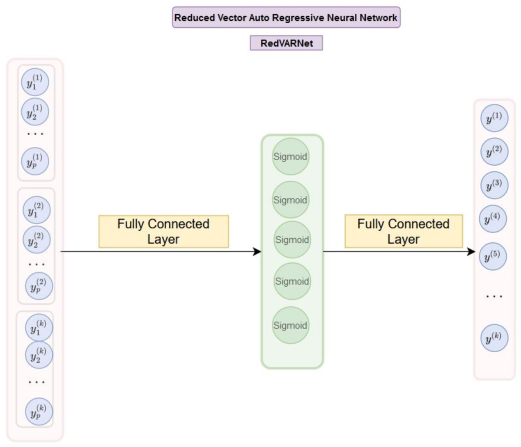


Figure 1: Reduced Vector Autorgressive Neural Network

models, thereby enabling them to jointly capture linear and non-linear dependencies at various scales. This dual capability significantly improves forecasting performance by addressing both short-term fluctuations and long-term trends.

The architecture of the **Reduced Vector Autoregressive Neural Network (RedVARNet)** is depicted in Figure 3.

Mathematically, let $Y_t = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_k(t) \end{pmatrix}$ denote the k-dimensional multivariate time series at time t. The inputs to the RedVARNet consist of the past p lags of the time series, represented as:

$$(Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}) = \begin{pmatrix} y_1(t-1) & y_1(t-2) & \ldots & y_1(t-p) \\ y_2(t-1) & y_2(t-2) & \ldots & y_2(t-p) \\ \vdots & \vdots & \vdots & \vdots \\ y_k(t-1) & y_k(t-2) & \ldots & y_k(t-p) \end{pmatrix}.$$

This input matrix is processed through a **fully connected layer** that performs a linear transformation on the inputs:

$$H = \sigma(W_1 \cdot \text{vec}(Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}) + b_1),$$

where W_1 is the weight matrix, b_1 is the bias vector, $\sigma(\cdot)$ denotes the non-linear activation function, and H represents the outputs from the hidden layer.

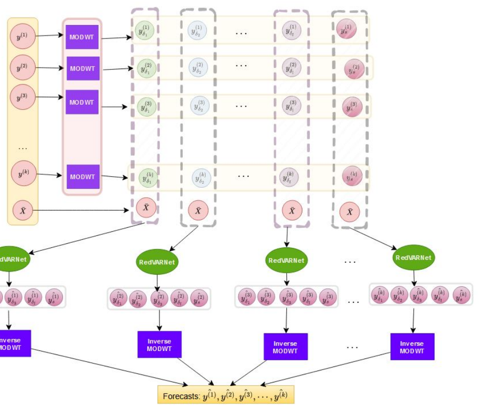


Figure 2: MEWNet

The outputs from the hidden layer are then forwarded through another **fully connected layer**, producing forecasts for the future values of the time series:

$$\hat{Y}_{t+h} = W_2 \cdot H + b_2,$$

where W_2 is the weight matrix for the second layer, b_2 is the bias vector, and $\hat{Y}_{t+h} = \begin{pmatrix} \hat{y}_1(t+h) \\ \hat{y}_2(t+h) \\ \vdots \\ \hat{y}_k(t+h) \end{pmatrix}$ denotes the predicted future values of the time series after h steps.

The architecture effectively leverages the past values of multiple time series through fully connected layers, employing non-linear activations to learn complex temporal dependencies.

The wavelet-based architecture depicted in Figure 2 exemplifies a **Wavelet-Based Multivariate Forecasting Model** that integrates Modified Discrete Wavelet Transform (MODWT) with the RedVARNet architecture to enhance forecasting accuracy.

The operational steps are as follows:

Let $Y_t = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_k(t) \end{pmatrix}$ represent the k-dimensional multivariate time series at time t. Initially, we apply the **Modified Discrete Wavelet Transform (MODWT)** to each individual time series $y_i(t)$. The MODWT decomposes each series into multiple scales, effectively capturing both low-frequency (approximation) and high-frequency (detail) components. This decomposition can be mathematically expressed as:

$$y_i(t) = \sum_{j=1}^{J} a_i^{(j)}(t) + d_i^{(j)}(t),$$

where $a_i^{(j)}(t)$ denotes the approximation coefficients (low-frequency) at scale j, and $d_i^{(j)}(t)$ represents the detail coefficients (high-frequency) at scale j.

Following wavelet decomposition, the approximation and detail coefficients $a_i^{(j)}(t)$ and $d_i^{(j)}(t)$ are independently processed through the **RedVARNet** model. For each scale j, the model takes the multivariate wavelet coefficients as input and forecasts their future values:

$$\hat{a}_i^{(j)}(t+h) = \text{RedVARNet}\left(a_1^{(j)}(t), a_2^{(j)}(t), \ldots, a_k^{(j)}(t)\right),$$

where h represents the forecast horizon and $\hat{a}_i^{(j)}(t+h)$ are the predicted approximation coefficients at time $t+h$. In parallel, the detail coefficients are similarly forecasted:

$$\hat{d}_i^{(j)}(t+h) = \text{RedVARNet}\left(d_1^{(j)}(t), d_2^{(j)}(t), \ldots, d_k^{(j)}(t)\right).$$

Upon forecasting the future values of both the approximation and detail coefficients across all scales, the inverse MODWT is employed to reconstruct the forecasted time series $\hat{Y}_{t+h}$:

$$\hat{y}_i(t+h) = \sum_{j=1}^{J} \hat{a}_i^{(j)}(t+h) + \hat{d}_i^{(j)}(t+h),$$

where $\hat{y}_i(t+h)$ is the forecasted value of the i-th time series at time $t+h$, and the sum combines the predicted approximation and detail coefficients from all scales.

Ultimately, the forecasts for all variables $\hat{y}_1(t+h), \hat{y}_2(t+h), \ldots, \hat{y}_k(t+h)$ are obtained, constituting the forecasted multivariate time series at time $t+h$.