Time Series Forecasting – Week 1

Introduction

Time series forecasting is a new technology that aims to provide predictions of future values based on historical, time-stamped data. Analyzing historical data can help us understand future trends and predict the behavior of a particular variable in the future. Compared to standard machine learning techniques such as random forest and time delay neural networks, the time series forecasting method can extrapolate patterns outside the training data, outperforming most machine learning algorithms that can only learn patterns from the training dataset without considering the temporal relationships between data points. This is where time series forecasting techniques come into play.

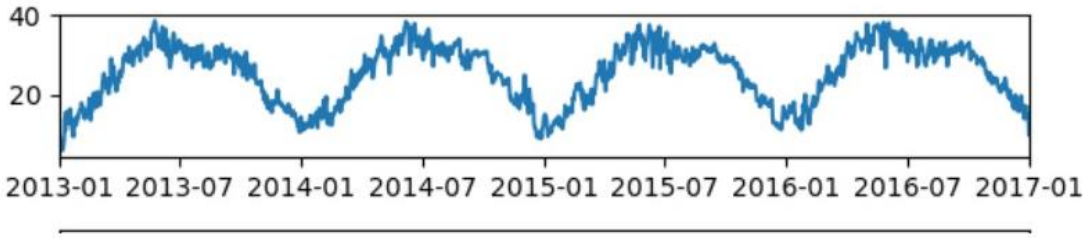
Time series forecasting is used in many industrial and scientific fields, including control engineering, business planning, disease spread modeling, signal processing, and weather forecasting.

For time series forecasting, three main terms are analyzed:

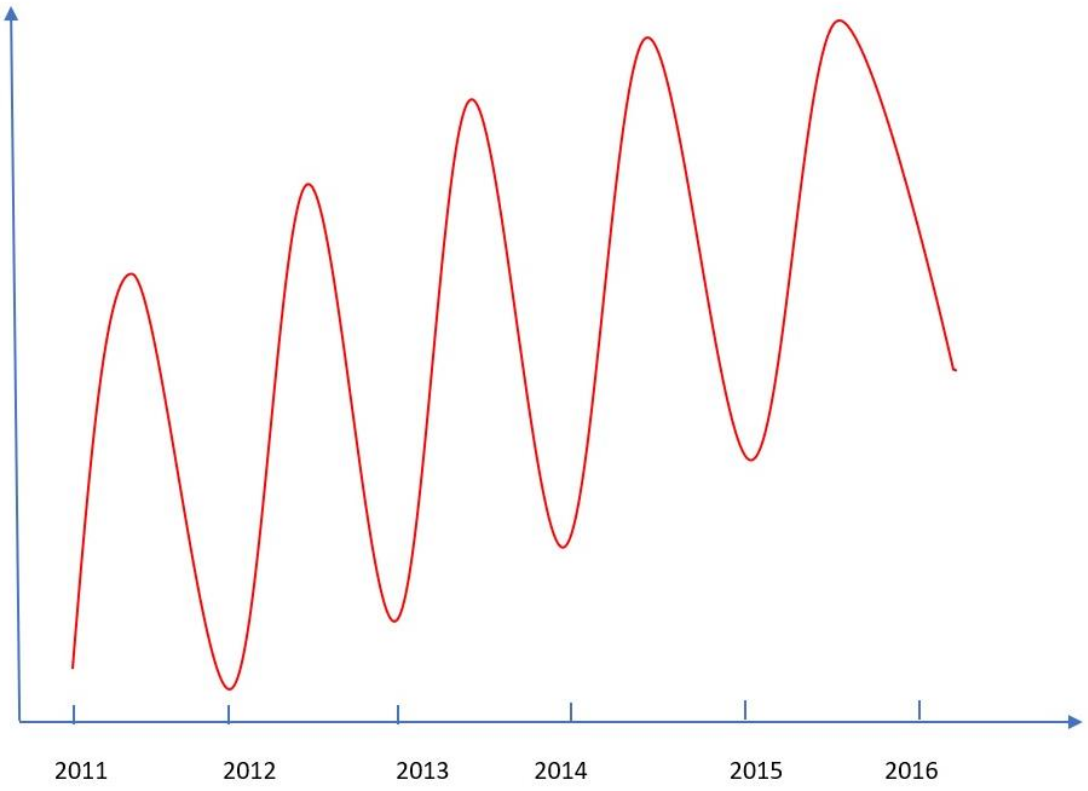
Seasonality:

Seasonality refers to the data patterns that repeat regularly in time series over certain periods of time, such as weeks or months. For example, we use the seasonality plot to observe peaks and falls in travel company data over a specific period in time. Time series Forecasting requires analyzing and capturing seasonality.

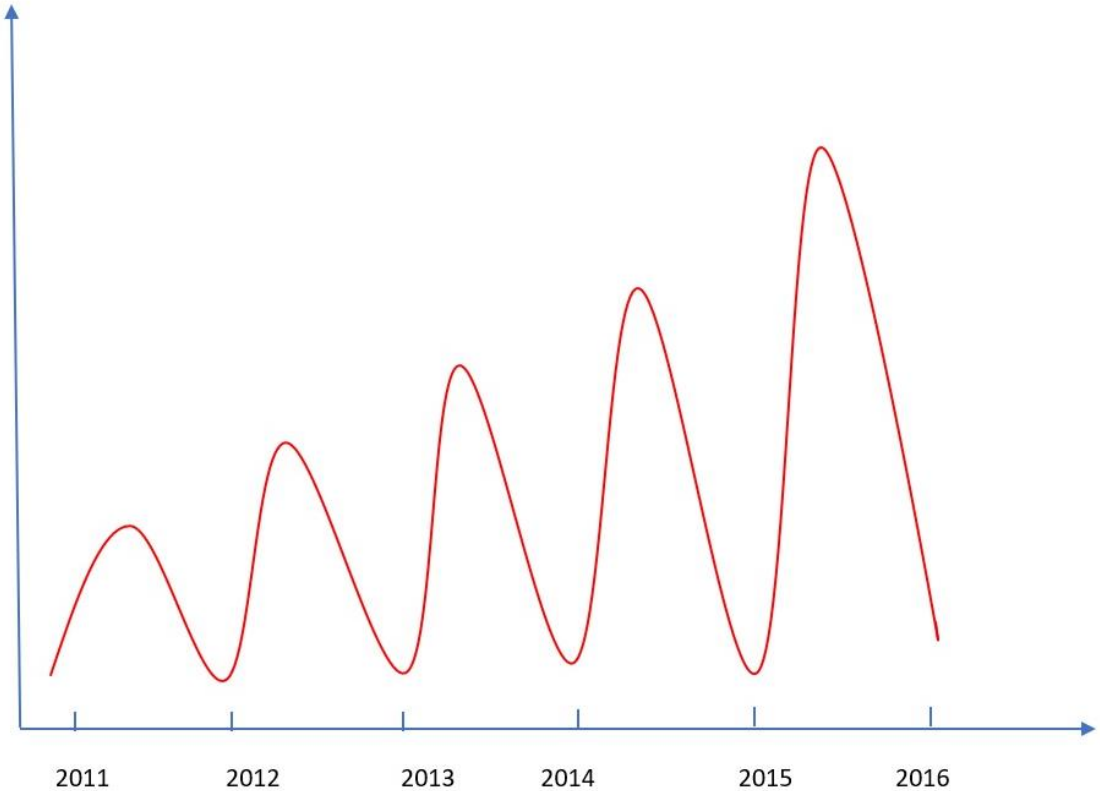
The following figure shows an example of an annual seasonal cycle:



We have two main types of seasonality: additive and multiplicative seasonality. In additive seasonal periods, the trend shows linearity, with the amplitude of seasonality remaining the same over time. In multiplicative seasonality, the changes in the width or height of seasonal periods and the trend is non-linear over time. The following figures explain the difference between the two types of seasonalities:



Additive Seasonality



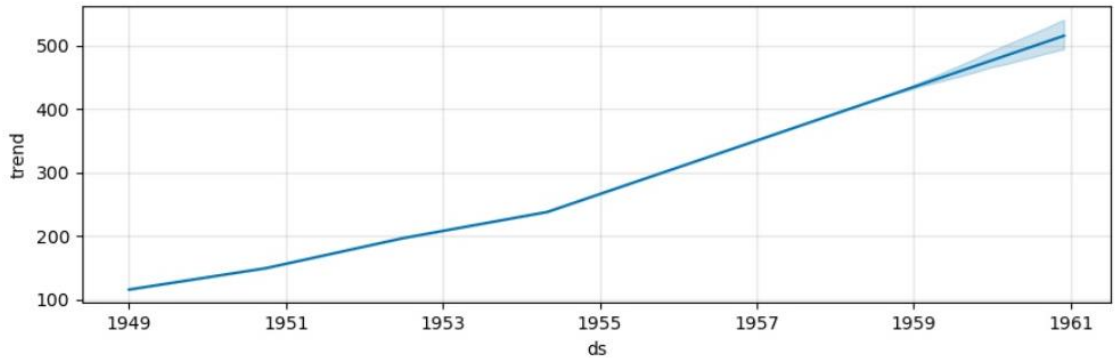
Multiplicative Seasonality

A

Trend:

Trend is another important factor in time series forecasting. It describes the increase or decrease of a variable over a certain period of time. For example, if a company's sales increase over a certain period of time, the sales variable shows an increasing trend.

The following plot depicts an example on increasing trend:



Cyclic Variation:

Cyclical variation describes fluctuations that recur over a period of time but not with a constant frequency. These rare changes in the data are called cycles and occur due to conditions with cyclical patterns, such as stock market fluctuations and seasonal temperature fluctuations.

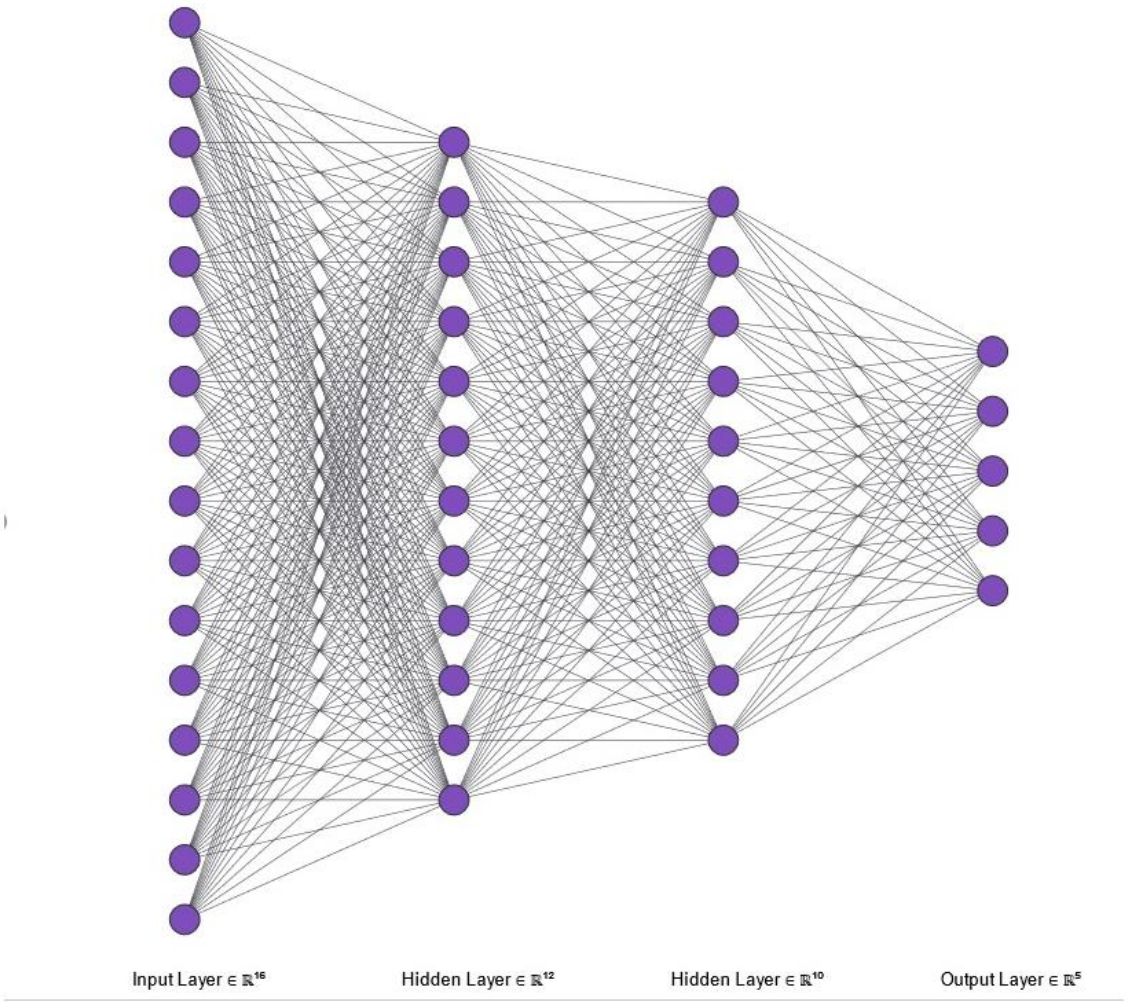
Irregular Variation:

This type of variation refers to the fluctuations that seem completely random or irregular and do not have a specific pattern. The fluctuations are unpredictable but must be taken into account when building time series models.

Introduction to Neural Networks + Long Short Term Memory (LSTM)

Artificial Neural Network (ANN)

ANN is a deep learning algorithm that processes data in a similar way to the human brain. It is often used to create and learn the pattern between the input features and the corresponding output in a dataset. The next figure shows the architecture of the neural network:



The first layer consists of a number of units n, so that for each input feature, it outputs n hidden states. The job of the hidden layers is to perform mathematical calculations on the inputs to produce the outputs that will be passed to the output layer. In a neural network for classification, the dimension of the output layer should correspond to the number of classes or labels in the training data. For example, a dataset with 3 classes requires a neural network to have 3 output neurons to learn the output pattern.

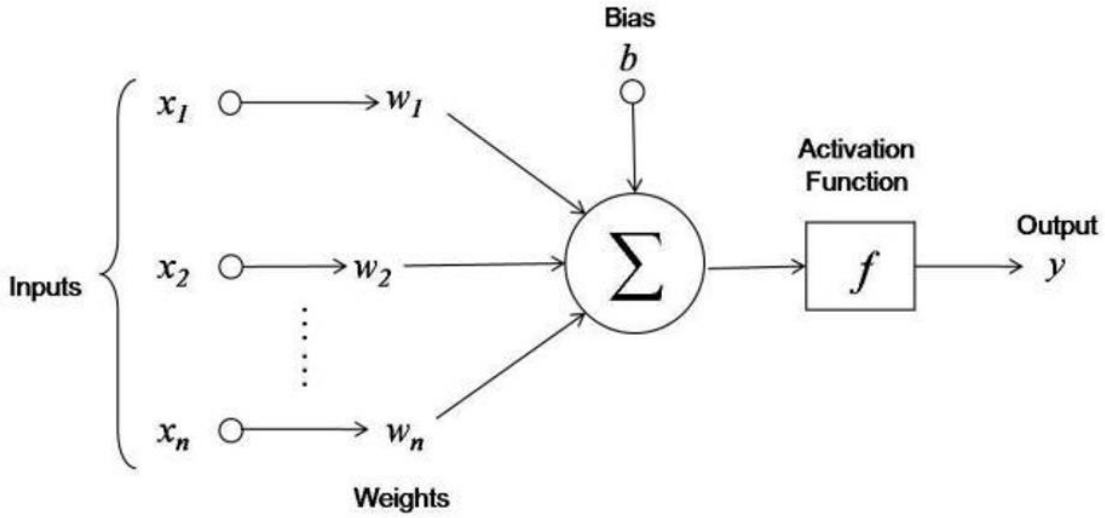
The neural network algorithm involves two steps:

1. Feed-forward neural network to pass data forward in the network.
2. Backpropagation of errors for changing the weights of the neural network during training phase.

1. Forward pass through a feed-forward neural network:

In this method, the information is passed through the hidden layers to the output layer of the network. The neurons of the first hidden layer receive a weighted sum of values of the input vector along with a bias term b, as shown in Fig. 2. Each

neuron then applies differentiable, nonlinear activation to this weighted sum to yield an output value. These steps are shown in the following figure:



Likewise, neurons of subsequent layers use a weighted sum of the outputs of all neurons of the previous layer along with a bias term as input. This process continues until the output layer neurons provide their output values. For a classification task, the output layer will calculate class probabilities, so the neuron with the highest probability value will decide the class of the input data.

In a neural network, each layer has its own activation function. For example, the activation functions used for ChatBot are the Rectified Linear Unit (ReLu) function for hidden layers and the Softmax function for the output layer. The ReLu function is defined as:

$$f(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

And the softmax function is defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where vector Z is raw logit vector from the neural network which can be thought of as predicted log probabilities of the input belonging to classes {1,..,K}. e^z is the exponential function of Z, K is the number of classes in the multiclass classifier. Softmax function is applied to convert the logits into probabilities from the range [0,1].

Given the raw output vector of neural network is Z={0.23, 1.7, 0.9} representing the predicted probabilities of classes {fruits, vegetables, seeds }, applying softmax on Z:

$Pr[y_i = \text{fruits}] = e^{0.23} / e^{0.23} + e^{1.7} + e^{0.9} = 0.08$
$Pr[y_i = \text{vegetables}] = e^{1.7} / e^{0.23} + e^{1.7} + e^{0.9} = 0.6$
$Pr[y_i = \text{seeds}] = e^{0.9} / e^{0.23} + e^{1.7} + e^{0.9} = 0.32$

2. Error back propagation to train a neural network:
This step is the core of training, as the main task of the neural network algorithm is to learn the correct set of weights for all layers that will lead to the correct output. It involves backpropagating the error or loss calculated at the output so that the new weights and biases are learned.

There are two main types of loss functions used in neural networks depending on the model type:

1. Regression loss functions: mean squared error, mean absolute error. Neural networks are used in regression.
2. Classification loss functions: Binary cross entropy, categorical cross entropy. This type of loss function is used for classification tasks.

Below are the formulas of MSE and cross entropy loss functions:

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(\hat{y}_i - y_i)^2$$

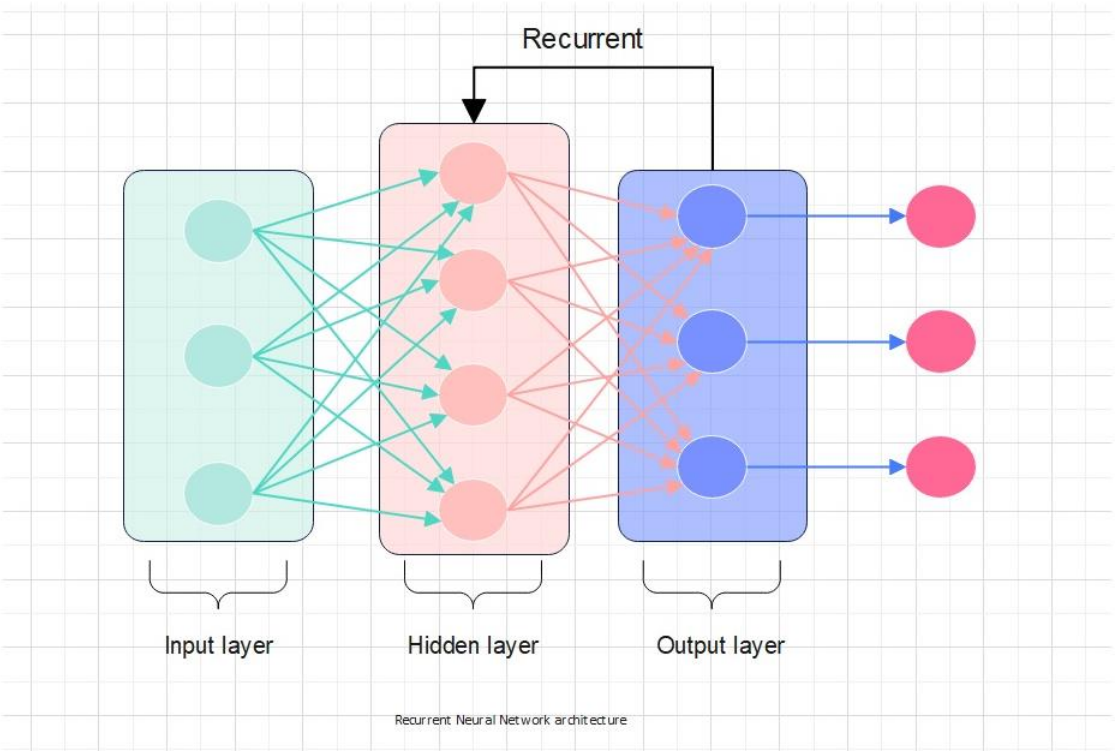
MSE is the average of the squared differences between the actual and predicted values. For multi-class classification, where the number of classes M > 2, categorical cross entropy is given by:

$$\text{CE loss} = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right)$$

$y_{o,c}$ takes 1 if the observation *o* is of class c, otherwise 0. $P_{o,c}$ is the predicted probability that observation *o* is of class c.

Recurrent Neural Networks (RNN):

Standard artificial neural networks are unable to capture the temporal dependence between samples. To incorporate such historical relationships, recurrent neural networks (RNN) have been adapted to predict future values by modeling past data. RNN is a variant of an artificial neural network equipped with a mechanism for dealing with sequential data or time series in which data points have a temporal correlation. The RNN architecture has a “memory” that helps saving the states of previous inputs and use them to calculate future predictions of the sequence.



LSTM:

The assumption of a linear relationship between inputs and outputs does not necessarily hold for every time series ,and hence the need arose for algorithms that can approximate any nonlinear function for time series forecasting without prior knowledge of the characteristics of the data. Long Short-Term Memory (LSTM) is one of the most popular time series forecasting models that has demonstrated its ability to make future forecasts using historical data. Thanks to long-term memory, LSTM is able to efficiently learn even more parameters and long-term trends in the time series.

LSTM networks are a special form of RNN architecture that is intended to better regulate the flow of information through the network. The main problem with RNNs lies in their limited ability to learn longer-term dependencies. Standard RNNs cannot learn if there are time delays of more than 5 to 10 discrete time steps between relevant input and target variables due to a “vanishing” or “exploding” gradient. A vanishing gradient occurs when the value of the product of the derivatives decreases until at some point the partial derivative of the loss function approaches a value close to zero, reducing its ability to learn long-term relationships. LSTM networks address this problem by enforcing a constant flow of errors using Constant Error Carrousels (CECs) in the memory cells. An LSTM memory cell contains input and output gates that control the error flow in the CEC. The multiplicative input gate protects the CEC from interference with irrelevant inputs, while the multiplicative output gate protects other units from interference caused by currently irrelevant memory contents. These gates allow the network to learn long-term relationships between data points more effectively.

Thanks to its lower sensitivity to temporal gaps, LSTM networks are more suitable for sequential data analysis than simple RNNs. In addition to the hidden state of RNNs, the architecture for an LSTM memory cell consists of , an input gate, an output gate, and an forget gate, as shown below:

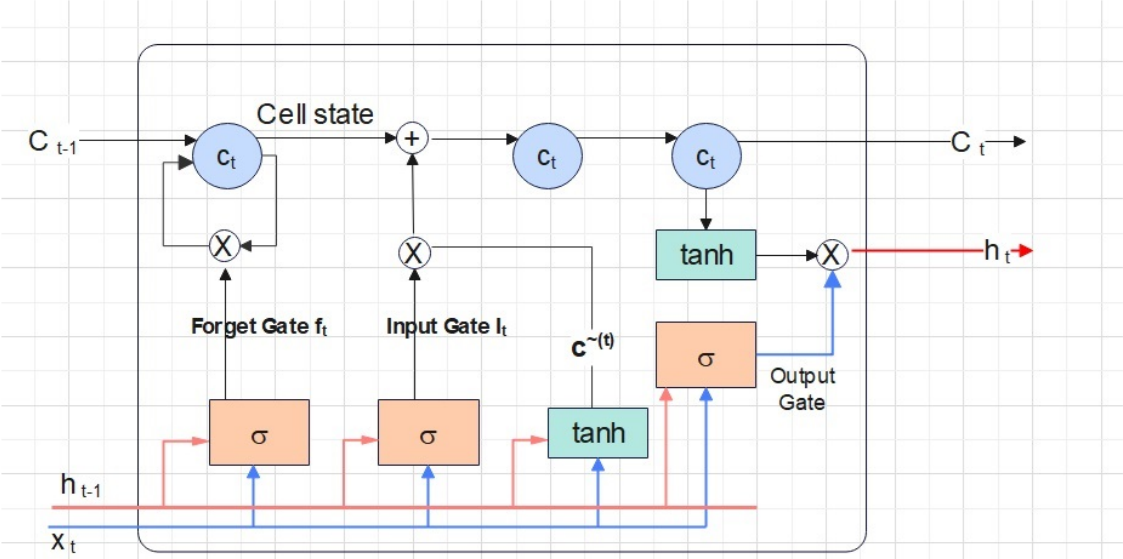


Figure: Long Short Term Memory (LSTM) cell (*)

Where h is the hidden state representing short-term memory, c is the cell state representing long-term memory, and x is the input. The input gate weights and biases control the extent to which a new value flows into the cell. Similarly, the weights and biases of the forget gate and the output gate control what data remains in the cell and what data is used to calculate the output activation of the LSTM block.

Exponential Smoothing

Exponential smoothing is a forecasting method for univariate time series data. Unlike the moving average method, which gives all past observations equal weights as they fall within the moving average window, the exponential smoothing method uses weights that decrease exponentially as the observations get older, removing outliers or noise from the data. This helps the model better identify distinct and repeating patterns that would otherwise be hidden in the noise and can significantly contribute to forecasting performance by improving the accuracy of predictions.

The different types of exponential smoothing include simple exponential smoothing and triple exponential smoothing (also known as the Holt-Winters method). Simple exponential smoothing deals with data series that have neither a trend nor a seasonal component. This method uses only one parameter called alpha (α) which determines the degree of smoothing and takes values between 0

and 1. Lower values result in smoother fitted models as more weight is given to the past data, thus efficiently averaging current with old data. Conversely, high values can produce a fitted model that responds quickly to the random noise. Simple exponential smoothing is represented in the following mathematical formulas:

$f_0 = x_0$

$f_t = \alpha\, x_{t-1} + (1- \alpha)\, f_{t-1}$

Where x_{t-1} is the observation in the previous time step. Alpha is the percentage of how important the most recent observation is compared to historical data in the model. f_{t-1} is the model's prediction in the previous time step.

The above equation can be translated into the following formula:

$$f_t = \alpha\, x_{t-1} + (1- \alpha)\, (\alpha\, x_{t-2} + (1 - \alpha)\, f_{t-2}) =$$

$$\alpha\, x_{t-1} + \alpha\, (1- \alpha)\, x_{t-2} + (1- \alpha)^2\, f_{t-2} =$$

$$\ldots\ldots\ldots$$

$$= \alpha\, [x_t + (1- \alpha)\, x_{t-1} + (1- \alpha)^2 x_{t-2} + \ldots + (1- \alpha)^{t-1}\, x_1] + (1- \alpha)^t\, x_0$$

From this equation, we can understand the main concept of exponential smoothing, where the model assigns exponentially lower weights to further observations than the most recent ones. Hence, the alpha value should be fine-tuned to get a well-fitting forecasting model. For example, simple exponential smoothing is used to model the demand for a product.

Triple exponential smoothing (Holt-Winters) is used for time series that exhibit trend and seasonal variations. It is an extension of simple exponential smoothing that allows the inclusion of seasonality and trend components to learn any hidden patterns in the time series and produce more accurate future forecasts.

Time Series Forecasting – Week 2

Autoregressive Integrated Moving Average (ARIMA)

The autoregressive integrated moving average (ARIMA) is a statistical method that uses an autoregressive model to derive future forecasts from time series based on historical data. ARIMA combines two main methods: an autoregressive model and a moving average model. The autoregressive method learns a linear equation between current and past values. The learned equation is used then to calculate future forecasts of the target variable. An autoregressive model of order p AR(p) is

given by:

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

Where ε_t is the white noise at time t, ϕ are the coefficients of the linear equation. P is the order of an autoregression and represents the number of past time steps that are incorporated when making forecasts.

On the other hand, a moving average model uses a linear combination of current and previous white noise error terms for its forecasts. The following equation explains a moving average model of order q:

$$y_t = C + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots\ldots + \theta q\, \varepsilon_{t-q}$$

Where θ is the coefficient of the linear equation.

When differentiating between consecutive observations is done together with autoregression and a moving average model, we get a non-seasonal ARIMA model. The ARIMA model can be represented by the following equation:

$$y'_t = C + \phi_1 y'_{t-1} + \ldots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \ldots + \theta q \varepsilon_{t-q} + \varepsilon_t$$

where y'_t is the differenced series.

This model is called ARIMA model (p, d, q) where p is the order of the autoregressive component, d is the order of the differencing involved and q is the order of the moving average. Differencing is a transformation method used to remove the temporal dependence in the time series. This involves eliminating components such as trends and seasonality to make time series stationary. The number of times the differencing is performed is called the order of differencing. So, first order differencing is the difference between the current and previous time data points ($y_i - y_{i-1}$) while second order difference is described as ($y_i - y_{i-1}$)-($y_{i-1} - y_{i-2}$). Since differentaition is an important step in preparing data for use in an ARIMA model, it must be done carefully to avoid over-differentiating the series. The correct differencing order is the minimum number of differencing steps required to obtain a nearly stationary series. For example, if the series still has positive autocorrelations for many lags (10 or more), the series needs to be further differenced. On the other hand, if the autocorrelation at lag 1 is too small, we may need to decrease the differencing order.

The SARIMA (Seasonal ARIMA) model is an extension of the ARIMA model. This is achieved by adding a seasonal terms to the equation. The seasonal part of the model consists of terms that are similar to the non-seasonal components (p,d,q), but involve backshifts of the seasonal period. SARIAM's equation is described by multiplying the additional seasonal by the non-seasonal terms.

Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components (TBATS)

While most common models like ARIMA and exponential smoothing can only learn single seasonality pattern, TBATS forecasting method is designed to model time series that include complex and multiple seasonal patterns by using exponential smoothing, for example daily data that may have a weekly and yearly pattern.

TBATS is the abbreviation for:

- Trigonometric Seasonality.
- Box-Cox-Transformation.
- ARMA-error.
- Trend.
- Seasonal Components.

In TBATS, a Box-Cox transformer is used to transform time series so that the data has a normal distribution. This is useful because many statistical methods assume a normal distribution of model errors. TBATS is then modeled as a linear equation of an exponentially smoothed trend, ARMA errors, and seasonal components modeled by trigonometric functions over Fourier series. ARMA errors are the errors (εt) from the ARIMA model. This enables TBATS to produce more accurate long-term forecasts and outperform other forecasting models. On the other hand, TBATS model shows lower prediction and learning performance when modeling long time series.

$$y_t^{(w)} = l_{t-1} + \phi b_{t-1} + \sum_{i=1}^{T} s_{t-1}^{(i)} + d_t$$

Where $y^{(w)}_t$ is used to represent the Box and Cox transformed observation with Parameter w, y_t is the observation at time t, l_t is the local level in the period t, b_t is

the short-term trend in the period *t*, *T* is the number of seasonal periods, $s^{(i)}_{t-1}$ represents the seasonal component at time *t-1* for the seasonal period *i*, d_t can be an ARMA model (p, q) that should model a Gaussian white noise process e_t with zero mean and constant variance .(*)

The local level model is a linear regression that models the unobserved components and stochastic trends in the time series.

Multivariate Time Series Forecasting

A multivariate time series consists of several interdependent variables recorded over time, where each variable may have different sampling frequencies and periodicities. In multivariate time series forecasting, the model focuses on the temporal dependency of a variable not only on its historical data but also on the past values of other variables.

Multivariate time series forecasting has applications in many fields such as energy, aerology, meteorology, finance, transportation, etc. One of the most commonly used methods for multivariate time series forecasting is vector autoregression (VAR). VAR generalizes a model by learning linear regression equation of variables. The equation is described by a linear relationship of each variable in the data set with the past values of itself and all other variables. The VAR equation for a variable *y* with *p* lags is given by:

$$y_t = a + w_1 * y_{t-1} + \ldots + w_p * y_{t-p} + \varepsilon$$

where *a* is a constant, $[w_1,\ldots w_p]$ is the weight vector, $[y_{t-1},\ldots,y_{t-p}]$ are the past values of *y* varaible, and *ε* is the error or white noise term. For all varaibles in the dataset, the previous equation is expressed as follows:

$$\begin{bmatrix} y1 \\ \vdots \\ yn \end{bmatrix} = \begin{bmatrix} a1 \\ \vdots \\ an \end{bmatrix} + \begin{bmatrix} w11 & \ldots & w1n \\ \vdots & \vdots & \vdots \\ wn1 & \ldots & wnn \end{bmatrix} \begin{bmatrix} y1(t-1) \\ \vdots \\ yn(t-1) \end{bmatrix} + \ldots\ldots + \begin{bmatrix} w'11 & \ldots & w'1n \\ \vdots & \vdots & \vdots \\ w'n1 & \ldots & w'nn \end{bmatrix} \begin{bmatrix} y1(t-p) \\ \vdots \\ yn(t-p) \end{bmatrix} + \begin{bmatrix} \varepsilon 1 \\ \vdots \\ \varepsilon n \end{bmatrix}$$

Where *P* is the number of past lags used to calculate the forecasts, *n* is the number of dependent variables, and *ε* is the white noise vector.

Time Series Forecasting – Week 3

Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is an decision tree ensemble algorithm that uses tree boosting technique to optimize the objective function by adding new trees to the model and fixing learning errors. XGBoost builds trees sequentially, with each new tree fixing errors of the previous ones, thus gradually improving the model's prediction performance. Adding trees to the model continues until error correction no longer improves the performance, which means the learning process is complete.

But how to learn the new decision trees?

as is always for all supervised learning models, we need to define and optimize an objective function which comprises loss and regularization functions as follows:

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i\ )^{(K)} + \sum_{i=1}^{K} w(f_i)\ \ (*)$$

Where *l* is the loss function, *w* is the regularization function, n is the number of samples or time sequences. At each step K, we have to add a new tree f_K that optimizes the objective function.

Both XGboost and Random Forests (RF) are ensemble methods that build multiple trees. The main difference between them is that XGBoost develops one tree at a time using all the data points, correcting errors introduced by previously trained trees, while Random Forest builds all the trees at once independently using a bootstrap sampling technique.

XGBoost is an effective method for time series forecasting although it requires transforming the time series into a supervised learning problem first. To evaluate the model, we need to apply walk-forward validation technique instead of k-fold cross-validation which would lead to optimistically biased results.

XGboost outperforms other algorithms due to the following capabilities:

Nonlinear relationships: XGBoost can handle complex, nonlinear relationships between input and target variables, allowing it to discover intricate patterns in time series.

Feature importance: XGBoost can identify the features that have the most and least influence on the target variable.

Regualrization: By utilizing Lasso regularization technique that applies a penalty to combat overfitting and improve the model accuracy, XGBoost ensures that the model generalizes well to new data.

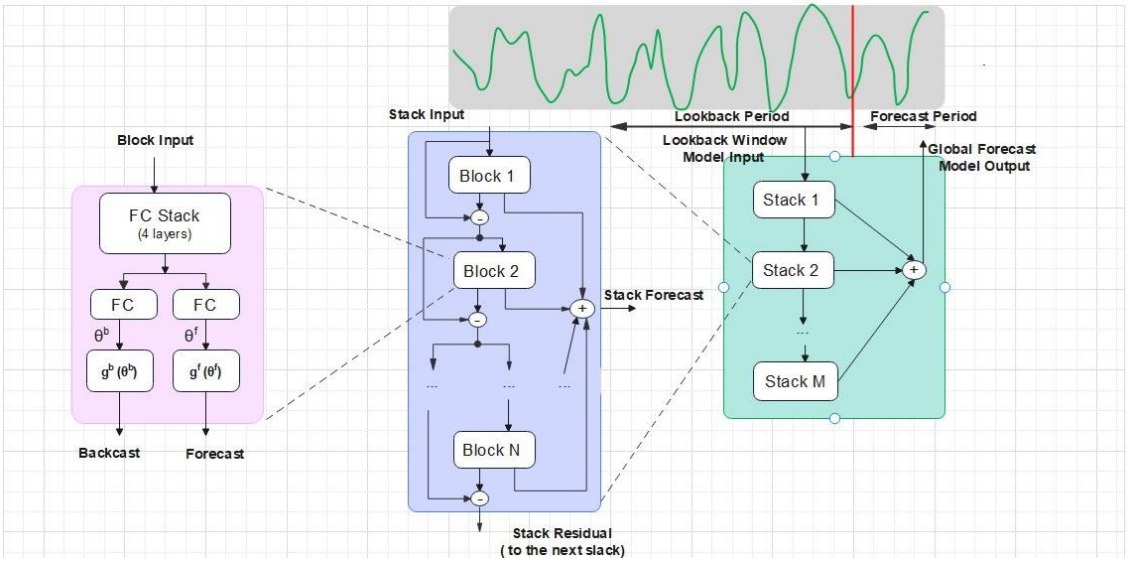
Dealing with missing values: XGBoost can handle missing values in the dataset, reducing the effort required for data preprocessing.

Neural Basis Expansion Analysis Time Series (N_BEATS)

Neural Basis Expansion Analysis Time Series is a deep learning architecture based on backward and forward links and a very deep stack of fully-connected layers. This architecture has proven to be effective and applicable in many areas where data is scarce. N_beats stands out for its fast training and interpretable results. In addition, double residual stacking enables a smoother loss backpropagation in the model.

Architecture:

The model consists of multiple stacks, each stack is in turn made up of a number of blocks. The model takes time series data up to x_t , also called lookback period, as its input and predicts future values up to $t+h$, where h is the forecast window length. N-BEATS architecture is illustrated in this figure:



N_BEATS Architecture

The input time series of size m is passed through four fully connected layers (see the figure below) of the first block. The output of the fully connected layers is fed

into two other FCs to produce a backcast vector of size *m*, and a forecast vector of size *n* as shown in this figure:

Doubly Residual Stacking Of Blocks:

A single Stack consists of multiple Basic Blocks, connected by Double Residual Stacking , where the term double indicates backcast and forecast outputs. The following figure depicts the structure of basic block:

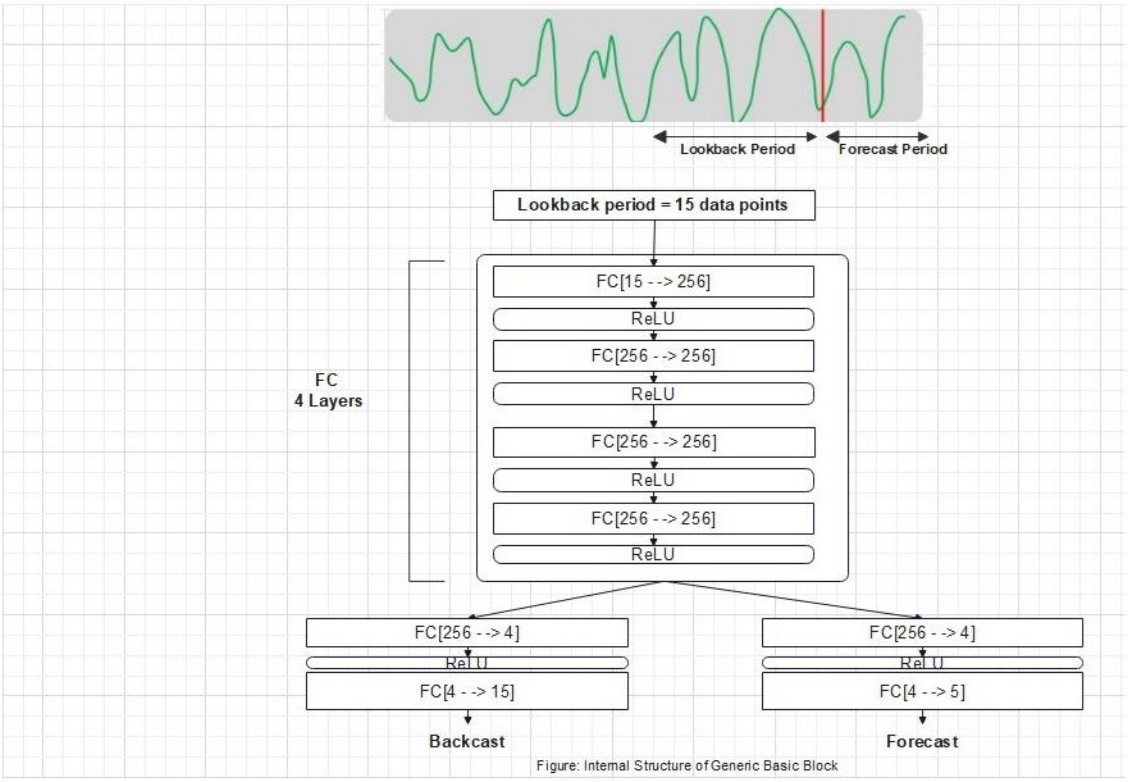


Figure: Internal Structure of Generic Basic Block

Assuming Backcast_1 is the output of block 1, the input time series will be then element-wise subtracted from Backcast_1, which yields an input vector of size m to block 2. This step ensures learning by forwarding loss values resulted from block 1 to block 2. In the stack architectue, only the first block gets the actual input sequence. While the following blocks get the residuals coming from the previous block. This means that only the information that was not captured by the one block is passed on the next. In other words, input to individual blocks will be the vector resulted from element wise subtraction of previous block's backcast and input values. The backcast output of the last block in a stack is called Stack Backcast output.

The forecast output of all the blocks in the stack will be element wise added to yield a vector of size n that represents a partial prediction of the input sequence.

Combining the Stacks:

Stack Backcast output from individual stacks will serve as input to next stack along the line, feeding loss into the model. Each stack uses a distinct function *g* (see N_BEATS Architecture above) that expresses specific components such as trend, seasonality, cyclic variation, etc. The partial forecast output from individual stacks will be element wise added to produce the final n-dimensional global forecast vector, which will be used to calculate the final loss value using the MSE (mean squared error) function. All the gradients in the architecture will be updated based on this loss.

Trend & Seasonality:

The output vectors of FC layer in the individual blocks X, Y will be multiplied by two polynomial coefficient matrices to learn trend and two Fourier coefficient matrices to learn seasonality. Those coefficient matrices are learned and adapted during training by the FC layer.

Facebook Prophet

The Prophet library is an open-source library designed for making forecasts using additive model by fitting non-linear trends with seasonality, while considering the effects of holidays. It is efficient for tasks where time series have strong and complicated seasonal patterns in the historical data. The Prophet equation is given by:

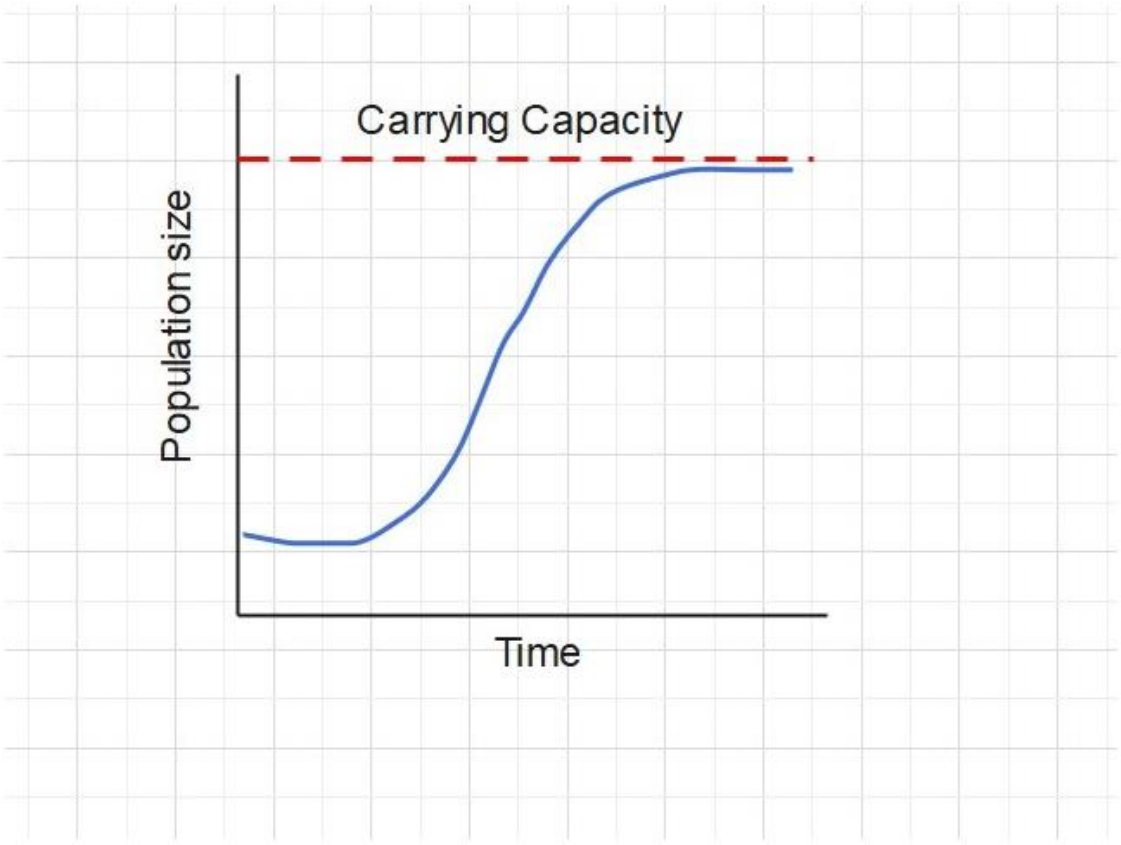
$$y(t) = g(t) + s(t) + h(t) + e(t)$$

where g(t), s(t), h(t), e(t) denote trend, seasonality, holiday and error terms respectively. Prophet model is trained on the time series to learn the coefficients of these four terms that lead to the most accurate predictions.

The trend function typically takes one of two forms: piece-wise linear model and logistic growth model. Choosing the right model is crucial to obtain a good forecasting model from the training data. If the data to be forecasted, has non-linear growing or shrinking trends with some seasonal changes, then logistic growth model is the best option. While the piece-wise linear model is a better choice if the data exhibits linear properties and trend behavior. The logistic growth model is given by the following equation (*):

$$g(t) = \frac{C}{1+e^{-k(t-m)}}$$

Where C is the carrying capacity, which refers to the limit or maximum value in the data at which the trend growth starts to decline and this is the point at which the forecast should reach saturation, k is the growth rate and m is the offset.



The piece-wise linear model is denoted as:

$$y = \begin{cases} \beta_0+\beta_1 x & x \leq c \\ \beta_0+\beta_2 c+ (\beta_1+\beta_2)x & x>c \end{cases}$$

where c is the trend change point.