**What is Time-Series ?**

* We know that when dealing with ML, the most important aspect we give considerable amount of importance is DATA !
* The bigger the volume of data, the more the chances grow for extracting interesting patterns and that galvanize us to dive deep and dig more to enhance the quality & quantity of meaningful questions as well as their answers – eventually, meeting a conclusion.
* When we talk about data – they depend on various factors – one such factor is **“TIME”**. During analysis, when we conclude that the data acquired changes/repeats after certain **time-intervals** OR the data has a considerable amount of dependence on **time** – then the data is called – **TIME SERIES DATA**
* Now let us discuss some important terms that explains the topic in a detailed fashion ---
  + **LEVEL :** To find the “level” of data (suppose you are given the data of SALES), the only action you will take is to find the **average**. Now average will be based on time.(e.g., in the SALES data, we can find the average sales value of 1 year – thus, time is included – the average value will be the **LEVEL** of data)
  + **TREND :** When we witness that – with time the results coming from data (as analysis Output or just values of a particular attribute) is increasing OR decreasing – we call this as “**TREND**”. Now trend are of 2 types – **Up-Trend** AND **Down-Trend (NOTE : the rate of increasing/decreasing of trend in reality is either steady, high OR low !).** Let us understand both the aforementioned terms through a real-life example. Adidas manufactures the Argentina FIFA World-Cup Football Jersey, and so is Nike for France. When the Final Match approached, the sales of Adidas & Nike apparels soared exponentially – This is called “**UP-TREND**”. On the other hand, as now FIFA World-Cup is finished , those apparels (Jerseys) are being bought at a lower frequency – this is called “**DOWN-TREND**”.
  + **SEASONALIY :** Let us take an example. We are taking the sales data of Air Conditioners throughout a year. After some analysis, we figure this out that during the summer season, sales of A/C units increase, but in the same year during the winter season , the units are bought in lower frequency – and it repeat every single year. What I mean to say is that, when we explore data – we see some interesting patterns as in increasing direction OR decreasing direction – which repeats over a fixed span of time (like here, say 1 year) – then we say that the data has **seasonality** feature.
  + **CYCLIC PATTERNS :** When you observe that the patterns in data are getting repeated after a longer span of time – then the patterns are called “**CYCLIC PATTERNS**”. **For ex. :** Lok-Sabha election are held every 5 years. When election are closing fast, the effect can be seen on stock prices – so this is a cyclic pattern of the data of stocks of various organizations.
  + **NOISE :** Sometimes, data do not show any pattern w.r.t. time. They rather show randomness in values. Those kinds of data are known as “**NOISE**”.

**Time-Series Analysis**

* Studying the features of the response variable with regard to time, as the independent variable, is done through “Time Series Analysis” (in short it is also known as TSA).
* TSA is the foundation for forecasting analysis and prediction, specifically for time-based issue statements like :
  + Examining the trends in the historical dataset
  + Recognizing and comparing patterns derived from the previous stage with the current circumstance.
  + Recognising the factor(s) impacting a certain variable(s) at various times.

**Time-Series Algorithms**

Now that we have a clear idea about “Time-Series” and data needed for “Time-Series Analysis”, it’s time for discussing some prominent algorithms which are unique in their approach but share the same functionality.

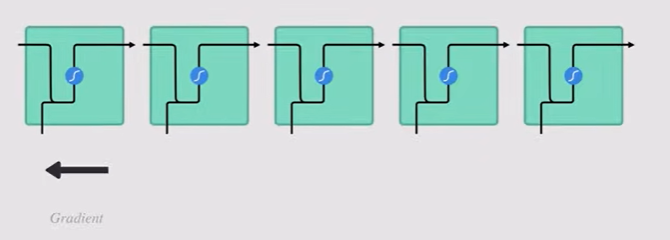
**Long Short-Term Memory (LSTM)**

* This algorithm evolved from its predecessor – Short Term Memory (also known as STM).
* STM has the problem of vanishing “Gradient”. To those who don’t know, “**Gradient”** is the value used to update a neural network’s weight.

**Gradient Update Rule**

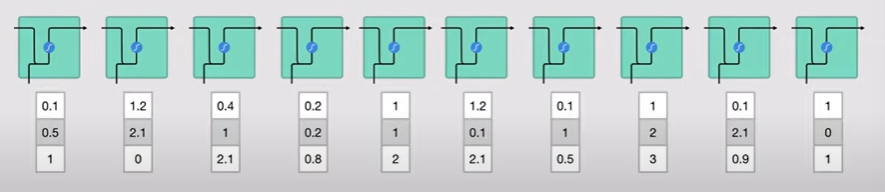
**New weight = initial weight – learning rate \* gradient**

The “vanishing gradient problem” is when a gradient strength as its back-propagates through time.

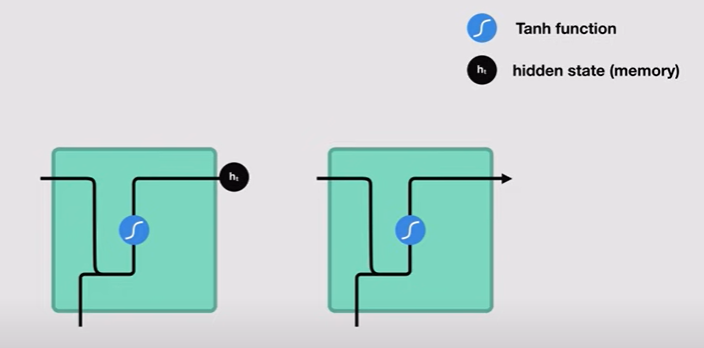


If a gradient’s value becomes very small, it won’t contribute much to learning. (Usually talking about the earlier layers). Hence, in longer sequences, the model won’t learn much (thus, having a short-term memory issue).

* Hence, LSTM was created as a solution to Short-Term Memory. Now, LSTM are mainly comprised of some internal mechanisms (also called **gates**) and some functions which are responsible to regulate the flow of data. These gates have the capability to learn which information to keep for future use and which ones to discard – for efficient learning.
* Below is a pictorial representation what STM does with the data it ingests –

****

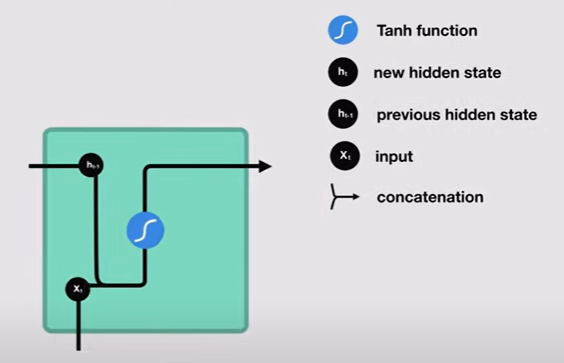
Suppose the model ingested a sentence, then each word is converted to machine readable vectors then sent each of the vectors one by one into the model’s function.

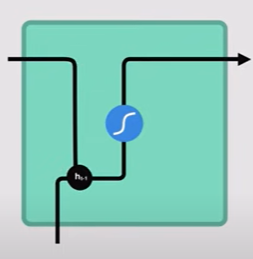
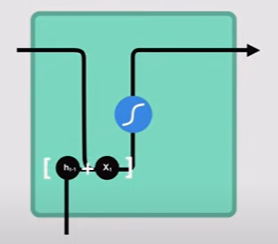


While processing, it passes the previous head state to next step of the sequence. The hidden state actually holds the information of neural network’s memory – what the model saw earlier.

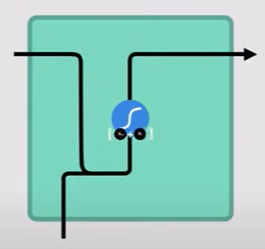
* Let us dive a little deep. Below is another picture which explains how the hidden state is calculated.

1. The input in the previous hidden state is combined to form a vector – which has information on the current inputs on the previous inputs.

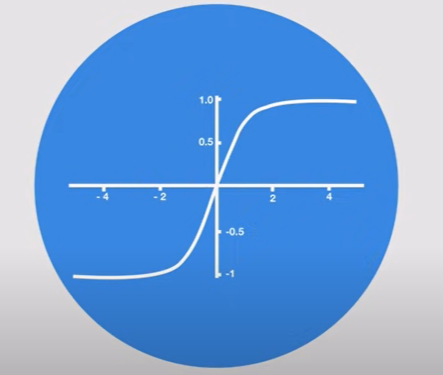


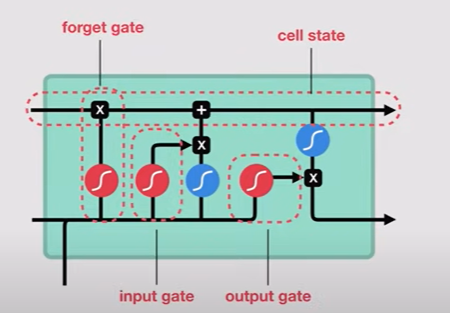
1. The vector goes through the **tanh** activation and the output is the new hidden state OR the memory of the network.



**tanh activation :** It’s used to help regulate the values flowing through the network. The function squishes values to always be between -1 and +1.

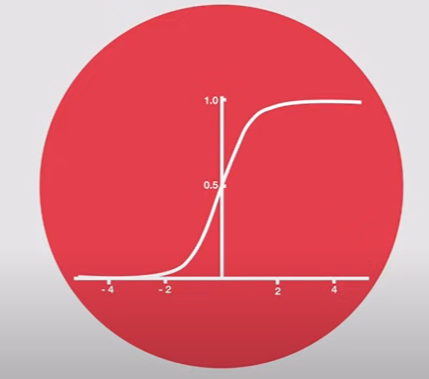


* If we keep on multiplying a set of numbers (let’s say (15, 0.03 , -5.44)) by a positive number (say 6) – some values can be very large, leading other values seem insignificant. On the other hand, the **tanh** function ensures that the values should lie between +1 and -1 – thus, regulated.
* Now let us talk about LSTM. Below is a picture that shows the insides of the algorithm –

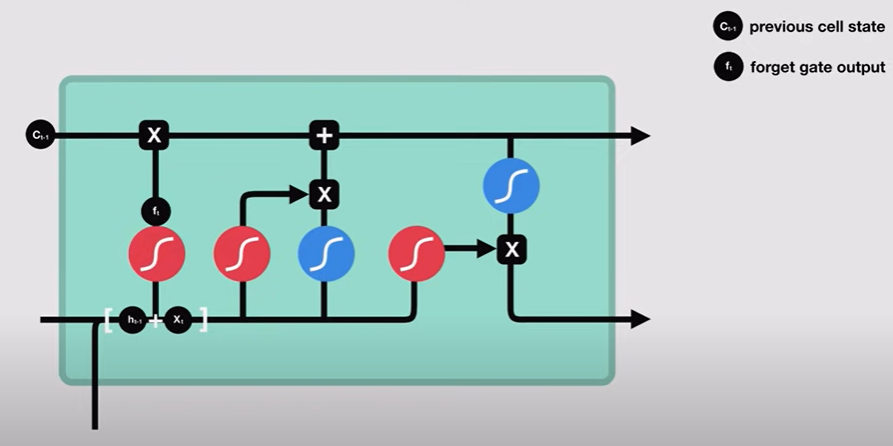


The control flow of LSTM is same as STM – ingests and processes data sequentially and passing on information as it propagates forward. So where is the difference between STM and LSTM ?

* The difference lies in the operations of LSTM Cells. These operations are used to allow LSTM to forget OR keep information.
* The cell state is like a highway, transferring relative information all the way down to the sequence chain (like memory of network) – from first to last time step (Thus, reducing the effects of STM). The dirtiness of added data keeps on getting cleaned because of the gates.
* Each gate themselves are different neural networks – which decides which information can be allowed to stay and which ones to discard during training. Gates contains **sigmoid activations** – which are similar to **tan** – the only difference – sigmoid squishes values b/w 0 and 1 , unlike tan (-1 and +1).

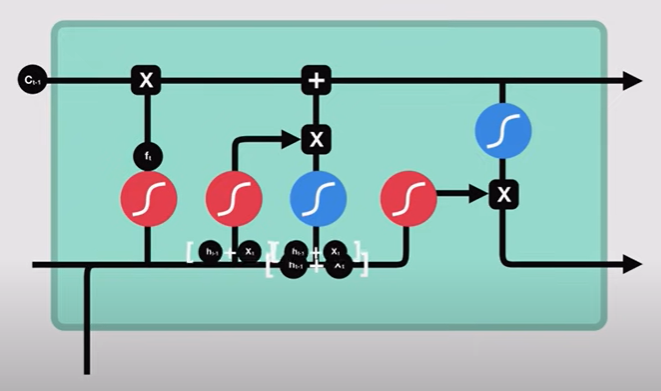


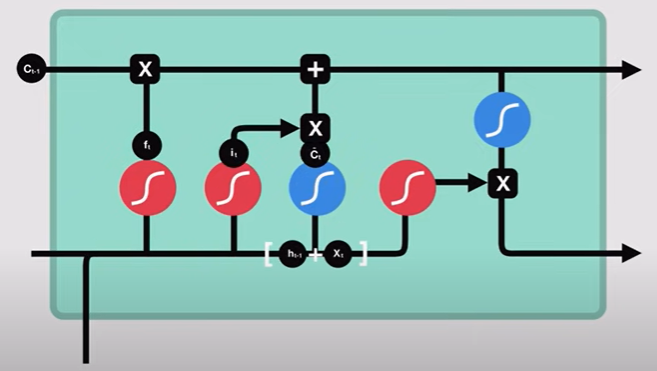
* ADVANTAGE : any number multiplied by 0 will be 0, so can be forgotten easily. On the other hand , value multiplied by 1 will remain the same (so, these will be the data to be kept).
* Now, there are 3 different gates – regulating the information flow. The gates are : **Input** gate, **Forget** gate and **Output** gate.
  + **FORGET GATE** : responsible for throwing away / disposing the information which are not required. Info. From previous hidden state and info. From the current i/p is passed into sigmoid function.



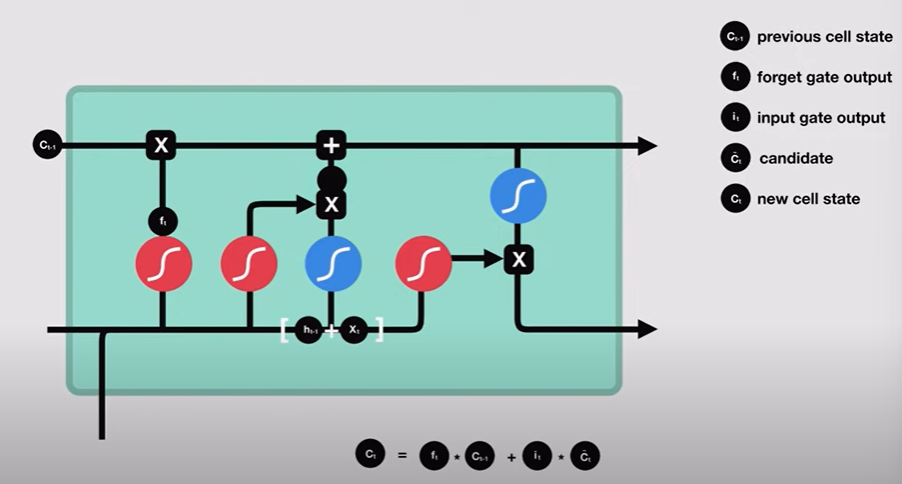
The value that will come up from the function will be either 0 or 1 (0 means forget, 1 means keep it).

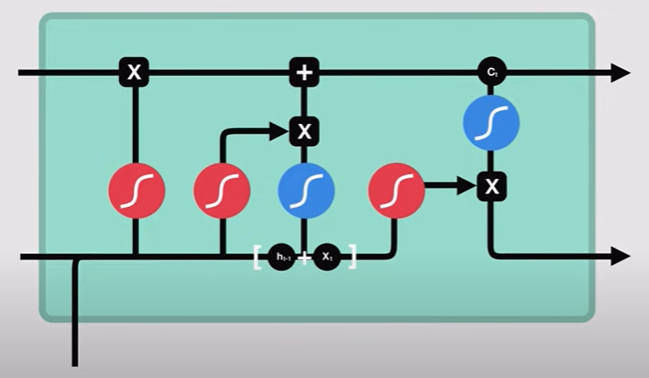
* **INPUT GATE** : It updates the cell state . First, we pass the previous hidden state and the current input to the sigmoid function, which decides which values will be updated – by transforming values to be between 0 and 1 (0 means unimportant, 1 means important). The hidden state & current input are also passed through the **tanh** function which will give out values b/w -1 and +1 to regulate the network.



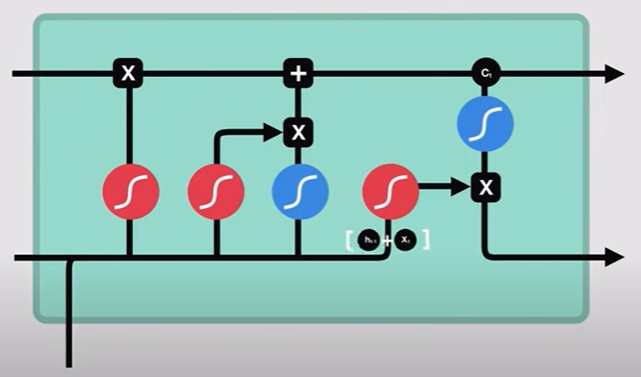


The output from sigmoid function and tanh function are then multiplied. The sigmoid output will decide which output from tanh function will be kept for the cell state –

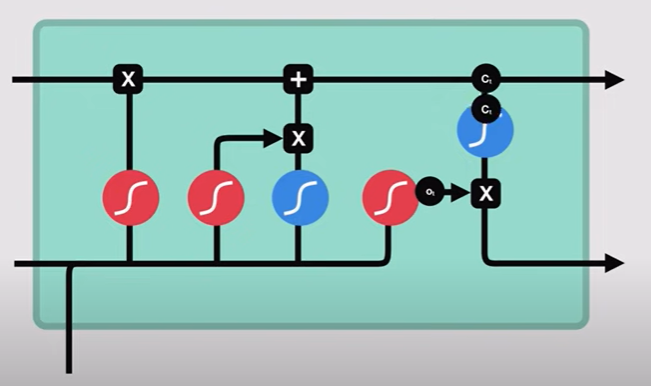




* **OUTPUT GATE :** This decides what the next hidden state should be. Remember that the hidden state contains info. Of previous inputs.

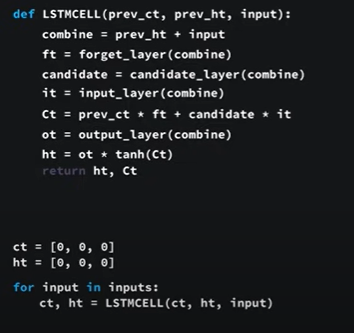


The previous hidden state and the current input are sent to sigmoid function and then pass the newly created/modified cell state to the tanh function.



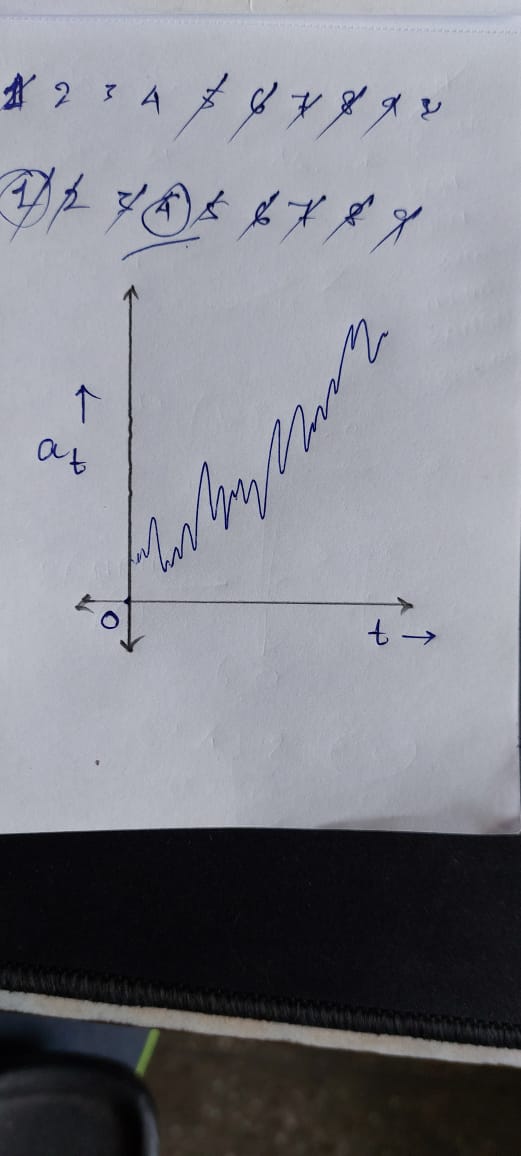
We then multiply the Sigmoid o/p with the tanh o/p and then decides what info. Should the hidden state (the output) carry and the cell state is continued to next stage of the sequence.

Below is the python code snippet of the algorithm of LSTM –



**ARIMA (Auto-Regressive Integrated Moving Average)**

Let us start the explanation using an example. Imagine that you are a salesman of “anchors” – for boats. Every month you sell a certain number of anchors – denoted by at (where ‘t’ denotes the time-period). Now you want to predict the number of anchors you are expecting to be sold next month. So, for analysis, you will be needing a graph – “at vs time” where at is on Y-axis (the output) and time is an X-axis (that keeps on going forward) – in the graph, you saw a pattern –



But the above data doesn’t seem to have depicting the quality of stationarity – which means that the time-series needs to have a constant mean, constant variance over time and has no seasonality. Here if you observe, you won’t see constant mean because the pattern seems like increasing.

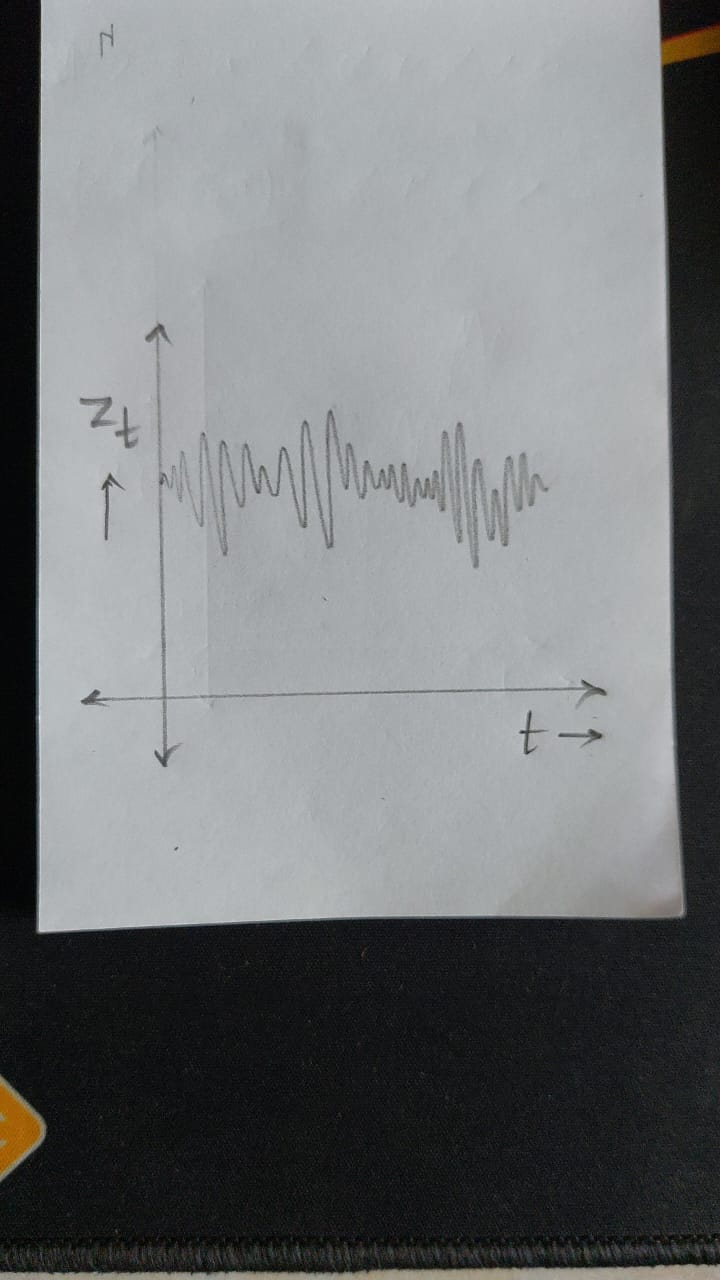
ARIMA solves the above issue as they can handle situations where everything is stationary except the **MEAN**. So here, instead of predicting time-series itself, we will be predicting differences of time-series from one time-stamp to the previous time-stamp – thus we are going to create a new time-series (called Zt) –

Zt = at+1 - at

Let us say that the above formula works like this – at+1 consists of anchor sold values of the month of February, and at denotes the same for January. Putting these values to the above formula will provide you with one time-point of the new series (known as Zt).

If you check the graph, it seems linear with some noise (as it’s not actually a straight line but we can make out the same by assuming it, that too inculcated with some noise). Now, for any linear function – if we want to go from one point to another – we just have to add a ‘constant’ value.

If it’s so then the difference between the current and the previous time-stamp should tending to be constant (may be hovering around some constant – see the below pic) –



Mean

Now that we have obtained a constant mean, thus the issue of the same is also solved – with no seasonality.

ARIMA has 3 parameters/orders : p,d,q . The ‘p’ is for AR (Auto-Regressive) , ‘d’ is for I (Integrated) and ‘q’ is for MA (Moving-Average). The simplest form of ARIMA is ARIMA (1,1,1) equivalent to ARIMA (p,d,q).

Now we are interested in finding out the difference between the consecutive time-points of anchor sales. So mathematically –

The formula shows the concept of Auto-Regressive (AR) components – as we are making out the function of a variable dependent on the previous value of a factor to get the output as the same variable dependent on the current value of the factor (here time).

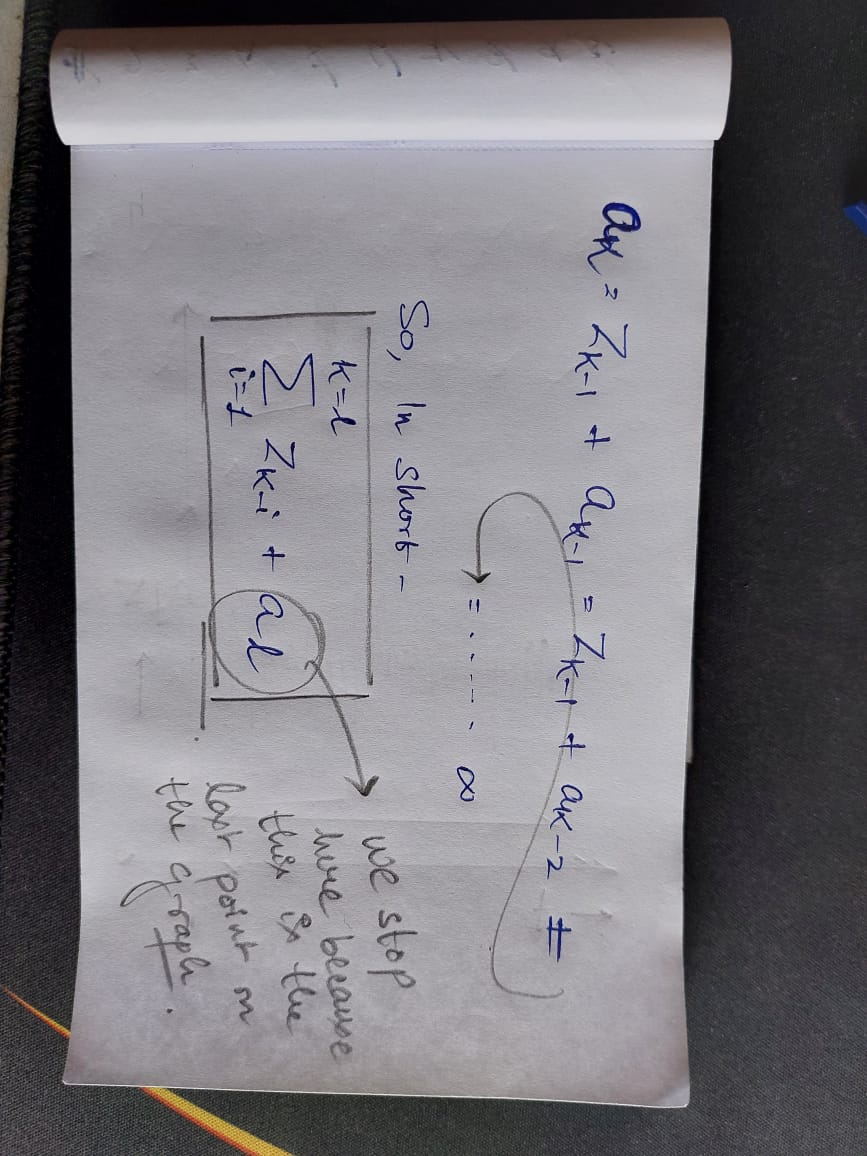
Zt = ɸ 1 \* Zt-1 + ⌀1\*ɛt-1 + ɛt

AR

MA

ɸ 1 is the coefficient. So, ⌀1\*ɛt-1 is the moving average (MA) bit and ɛt is the error in the current time period. The ‘Integrated’ component is taken care by Zt (the difference of consecutive points we originally started with).

But we are not winding up by predicting Zt – we are here to predict the no. of anchors we can sell (denoted by ak). Let us assume that the last point (t) of the sales graph be **t = l 🡪 al. So, to calculate ak –**

Thus, we will get to know how many anchors are sold (ak) in a certain time-period (k)- eventually will predict show many anchors will be sold in the next month.

--------