# Household Power Consumption Prediction using RNN-LSTM

Power outage accidents will cause huge economic loss to the social economy. Therefore, it is very important to predict power consumption.

Given the rise of smart electricity meters and the wide adoption of electricity generation technology like solar panels, there is a wealth of electricity usage data available.

# Problem Statement:

Given that power consumption data for the previous week, we have to predict the power consumption for the next week.

Dataset Description:

The data was collected between December 2006 and November 2010 and observations of power consumption within the household were collected every minute.

It is a multivariate series comprised of seven variables

* ***global\_active\_power***: The total active power consumed by the household (kilowatts).
* ***global\_reactive\_power***: The total reactive power consumed by the household (kilowatts).
* ***voltage***: Average voltage (volts).
* ***global\_intensity***: Average current intensity (amps).
* ***sub\_metering\_1***: Active energy for kitchen (watt-hours of active energy).
* ***sub\_metering\_2***: Active energy for laundry (watt-hours of active energy).
* ***sub\_metering\_3***: Active energy for climate control systems (watt-hours of active energy).

This data represents a multivariate time series of power-related variables that in turn could be used to model and even forecast future electricity consumption

Time-series predictions play a major role in machine learning which is often neglected. Nonetheless, there are lots of machine learning algorithms we could use for these problems. The major machine learning algorithms involving **Statsmodels**and **Econometric**models etc. Today we will take a look at how to use and apply Deep learning algorithms to predict the time series Data

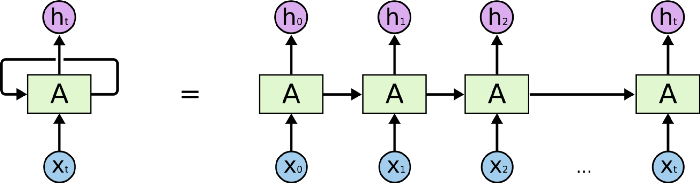
Why use a Deep Learning Algorithm?

With the data volume growing enormous day by day we shouldn’t confine ourselves to only the standard ML algorithms. *Deep learning algorithms* help us to handle large volumes of data and without leaving the key insights and by tuning the model within the right way gives us the maximum yield i.e., in our cause *maximum accuracy* 😊 . The model also determines if our prediction is better or worse from its own neural network architecture.

*For this Time series forecasting we will use Long- Short Term Memory unit (LSTM).*

Recurrent Neural Network (RNN)

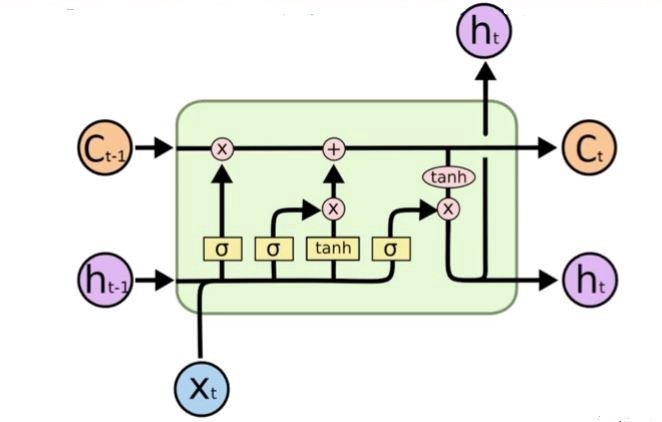
To understand an LSTM Network, we need to understand a Recurrent Neural Network first. This kind of network is used to recognize patterns when past results have influence on the present result. An example of RNN usage is the time-series functions, in which the data order is extremely important. In this network architecture, the neuron uses as input not only the regular input (the previous layer output), but also its previous state.



It is important to notice that **H**represents the neuron state. Therefore, when in state H\_1, the neuron uses as input the parameter X\_1 and H\_0 (its previous state). The main problem of this model is the memory loss. The network older states are fast forgotten. In sequences where we need to remember beyond the immediate past, RNNs fail to remember.

***Long Short Term Memory*** unit(**LSTM**) was typically created to overcome the limitations of a Recurrent neural network (**RNN**). The Typical long data sets of **Time series** can actually be a time-consuming process which could typically slow down the training time of **RNN**architecture. We could restrict the data volume but this a loss of information. And in any time-series data sets, there is a need to know the previous trends and the seasonality of data of the overall data set to make the right predictions.

Before going into the brief explanation of LSTM cell, Let us see how the LSTM cell looks like :



The Architecture may look little complicated on the first glance, but it is pretty neat and clear and easily understandable if we break it into parts.

Lets first start understanding what are our inputs and outputs. The typical input if you see on the left-hand side of the diagram Ct-1 which is the previous cell state and ht-1 which is the output from the previous cell and Xt which is the input of the present cell.

The output of the cell is Ct and ht which are the corresponding cell state and output of the present cell. The first step of an LSTM is the forget gate layer (f) where we determine what are we going to forget from the previous cell state. This typically takes the input ht-1 and Xt and make a linear transformation with some weights and bias terms and pass into the sigmoid function. As we are aware the output of a sigmoid function is always between 0 and 1. Here 0 will be considered as to forget it and 1 will represent to keep it

***Forget gate later=> f = Sigmoid ( Weights (ht-1,Xt) + bias)***

The second step is a two-part process and this is the step which tells us actually processing within this layer. Here in the first part we take the same inputs as before the ht-1 and Xt and make a linear transformation with some weights and biases and pass on to a sigmoid function. And the second part we will make a linear transformation again between ht-1 and Xt with some weights and biases but this time its going to be a hyperbolic tangent function (tanh). At the end of this step, we will get vectors of values which can be new candidate values for this present cell.

***First part => I = sigmoid( Weights (ht-1,Xt) + bias)***

***Second part => II = tanh( Weights (ht-1,Xt) + bias)***

The third step is the update step which helps us in deriving the new cell state Ct using our previous steps. First, we will multiply the previous cell state with the forget gate layer and add the vectors we got from the second step which forms the new cell state Ct of the present cell at t.

***Update layer => Ct = Ct-1 f + I II***

The final step is another main output of the cell, for this, we will directly form a linear transformation with the previous output ht-1 and input of the present cell Xt with some bias and weight terms and pass on to a sigmoid layer. Finally, now we will multiply this output to the new cell state Ct which is passed on to a hyperbolic tangent function. This gives us the present output ht.

Final layer =>

***i = sigmoid ( Weights (ht-1,xt) + bias)***

***final ht = i \* tanh(Ct)***

*Now we have a clear understanding of the step by step dissection of the LSTM layer. Let’s see how we apply our LSTM cell into a time series data.*