Domain Background:

Records of prices for traded commodities go back thousands of years. Merchants along popular silk routes would keep records of traded goods to try and predict price trends, so that they could benefit from them.

In finance the field of quantitative analysis is about 25 years old and even now it's not fully accepted,understood or widely used.Quantitative analysis is the study of how certain variables correlate with stock price behavior.

One of the first attempts at this was made in 1970 by two british statisticians named Box and Jenkins using mainframe computers. The only historical data they had access to were prices and volume.They called their model Arima and at the time it was extremely slow and expensive to run but by the 80s things started to get interesting. Spreadsheets were invented so that the firms could model company's financial performance and automated data collection became a reality. And with improvements in computing power, model could analyze data much faster, it was a renaissance on Wall Street, people were excited about the new possibilities .

Box and Jenkins work:
https://en.wikipedia.org/wiki/Box%E2%80%93Jenkins_method

Investor makes educated guess by analysing data, they read the news, study the company history , company trends and there are lot more points that go into stock price prediction. The prevailing theory is that stock prices are totally random and unpredictable.

As Burton Malkiel said,
" A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts"

But that rises a question that why do top companies like Morgan stanley and Citigroup hire a quantitative analyst to build predictive models? We have this idea of a trading floor being filled with adrenaline-infused man with loose ties running around yelling something into a phone. But these days we are more likely to see rows of machine learning experts quietly sitting in front of computers. Infact about 70% of all order on Wall Street are now placed by a software.
We are now living in the age of algorithms.

In the past few years we have seen lots of academic papers published using neural nets to predict stock prices with varying degrees of success. But until recently the ability to build these models has been restricted to academics who spend their days writing very complex code.Now

with libraries like Tensorflow, keras etc anyone can build powerful models trained on massive data sets.

Research papers on stock prediction using nn:
https://people.eecs.berkeley.edu/~akar/IITK_website/EE671/report_stock.pdf
https://nseindia.com/content/research/FinalPaper206.pdf

Motivation behind the project:

This project is about building a deep learning model to predict the stock prices. For this project I'll being using Keras framework with Tensorflow backend to predict the stock price of Google.

Problem statement :

To build a deep learning model using LST network to predict the stock price of Google using daily closing price of the S&P from Jan 2000 - Aug 2016 for training data. This is a series of data points indexed in time series.Our goal would be to predict the closing price for any given date after training

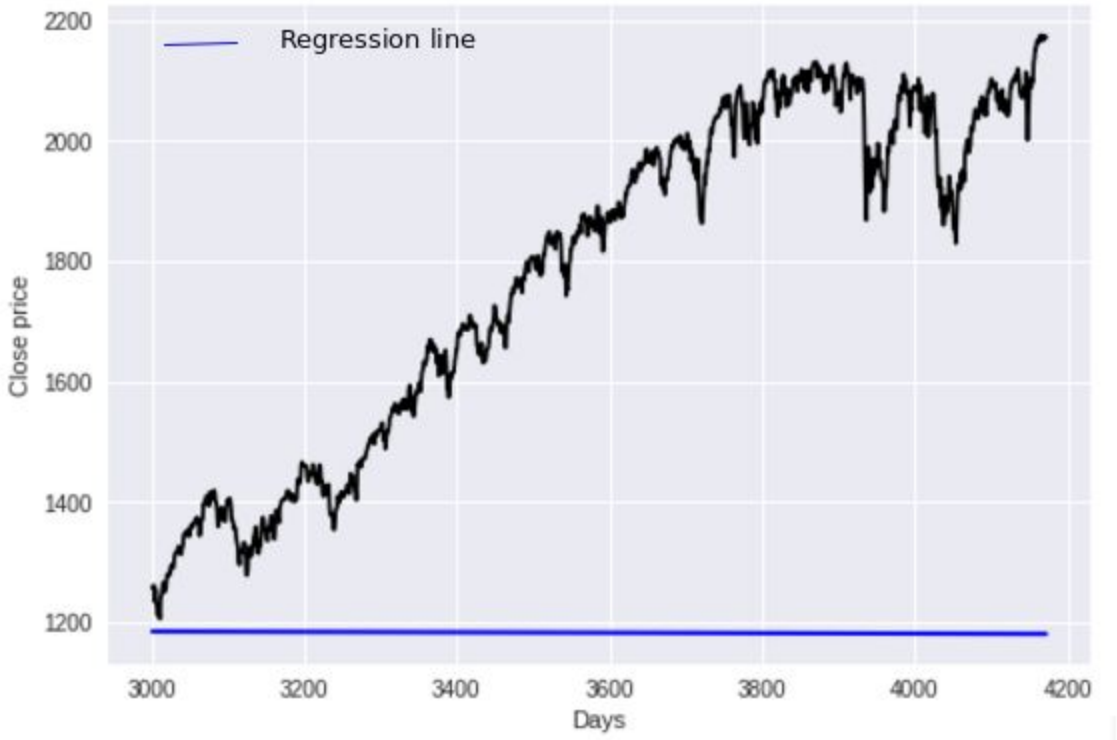
Data sets and inputs:

The data set consists of daily closing prices of Google from Jan 2000 - Aug 2016.Here the only feature that we are passing to the learning model is the closing price of the Google stock price on various days. These features can extracted from other google finance page.
Google FInance:
https://www.google.com/finance?q=google&ei=GJDnWJC4McaFuwTVgZrgCg

Dataset :https://drive.google.com/file/d/0B26anC5sNYXqTUxuTVlSaFlrSVE/view?usp=sharing

Benchmark model:

As a benchmark model I'll be using Linear Regression model against the deep learning model. In both cases error function or cost function and the error metric both is mean square error itself.



Linear regression model for this dataset can be found here: https://github.com/satishjasthi/Udacity-MLNDP-Submissions/blob/master/CapstoneProject/Linear.ipynb

Workflow of project:

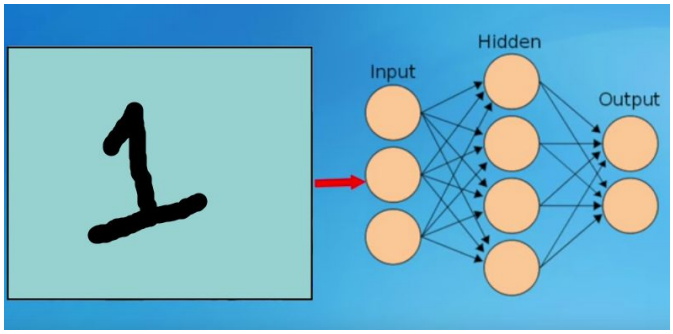
1.Data preprocessing: Initially the input data is normalised,rather than directly using the raw values normalised values helps in faster convergence of the algorithm. When our model predicts the values we'll then denormalise the data to get a real world number out of it.

2. To build our model we first initialize it as a sequence, it will be a linear stack of layers
Code:

Model = Sequential()

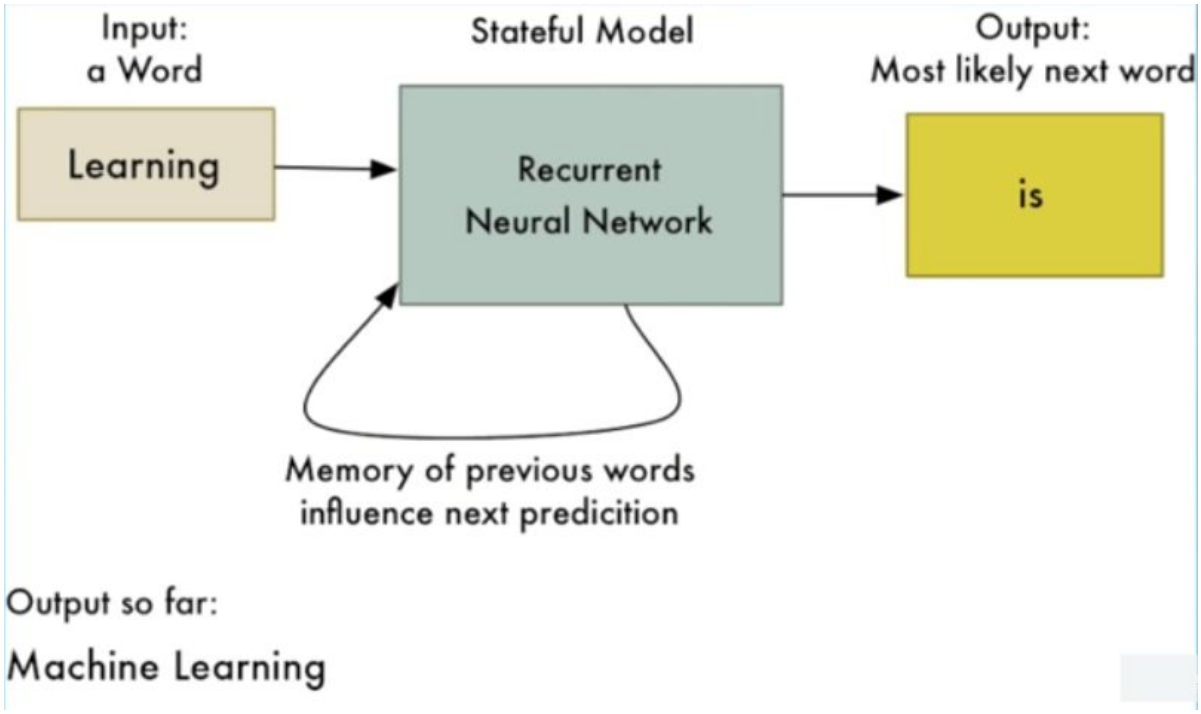
model.add(LSTM(input_dimension,output_dimension,return_sequence))

Why do we do this?
In case of sequences, memory matters because it uses conditional memory.However feedforward neural networks don't have this property. They accept a fixed size vector as an input, like an image.

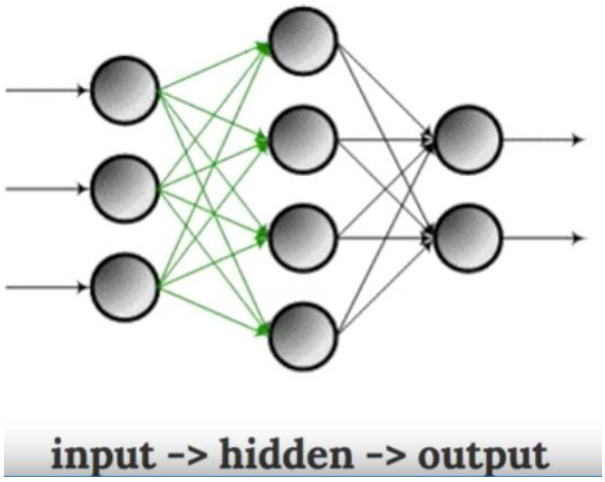


So we couldn't use it to predict the next frame in a movie because that would require a sequence of image vectors as inputs. And not just one since the probability of a certain event happening would depend on what happened every frame before it.

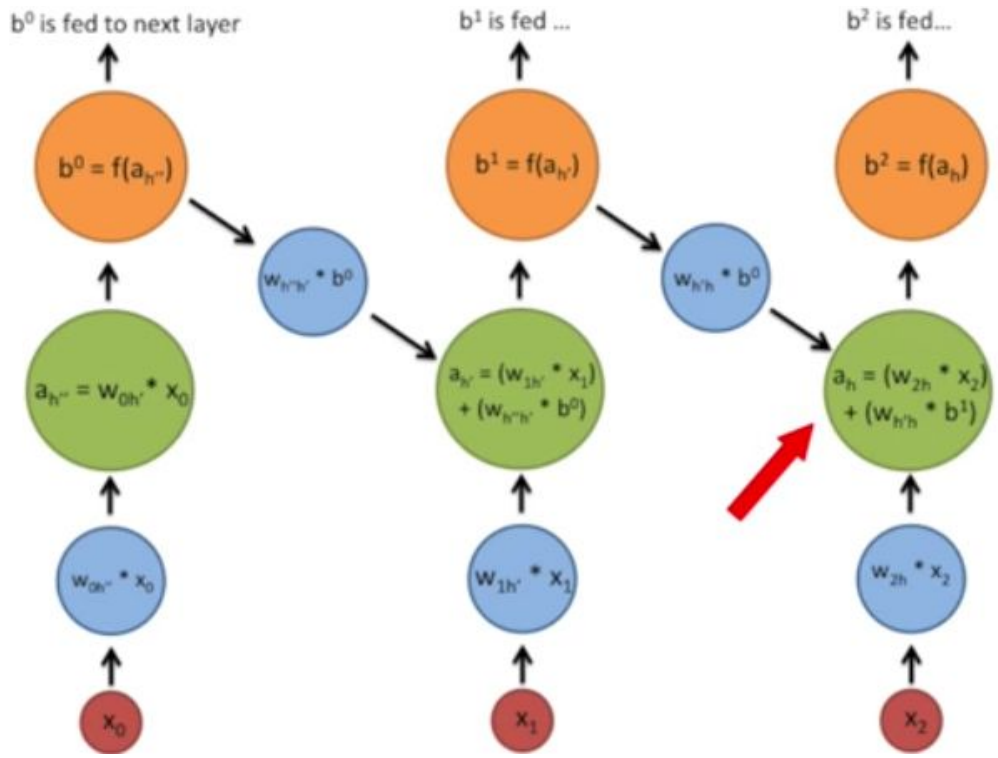
So we need a way to allow information to persist and that's why we can use a recurrent neural net. Recurrent neural nets can accept sequences of vectors as inputs.



So as we discussed before, in feedforward neural nets, the hidden layer's weights are based only on the input data.



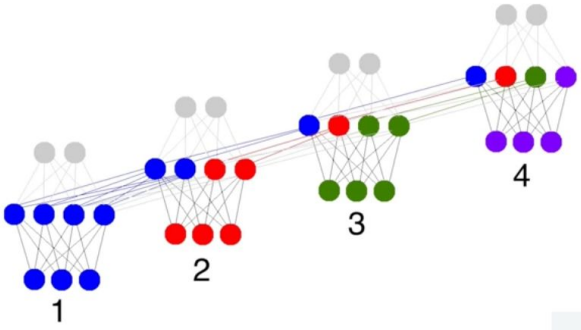
But in a recurrent net, the hidden layer is a combination of the input data at the current time step and the hidden layer at a previous time step. The hidden layer is constantly changing as it gets more inputs and the only way to reach these hidden states is with correct sequence inputs. This is how memory is incorporated in and we can model this process mathematically as



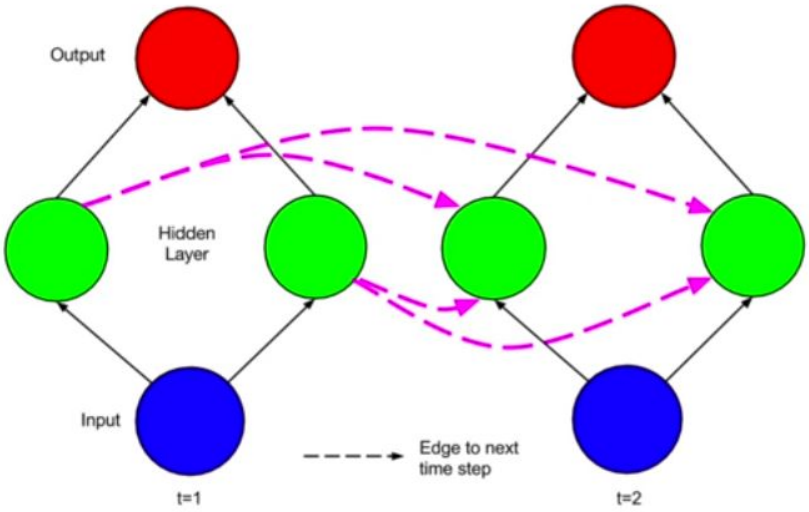
ht = ϕ (W * Xt + U * h(t-1))

Where ht = hidden state at a given time step
Xt = input at the given time step
W = weight matrix similar to one used in feedforward neural nets
ht-1 = hidden state of previous time step
U = transition matrix

And because this feedback loop is occurring at every time step in the series each hidden state has traces of not only the previous hidden state but also of all of those that preceded it. That's why we call it Recurrent Neural net.



In a way, we can think of it as copies of the same network each passing a message to the next.



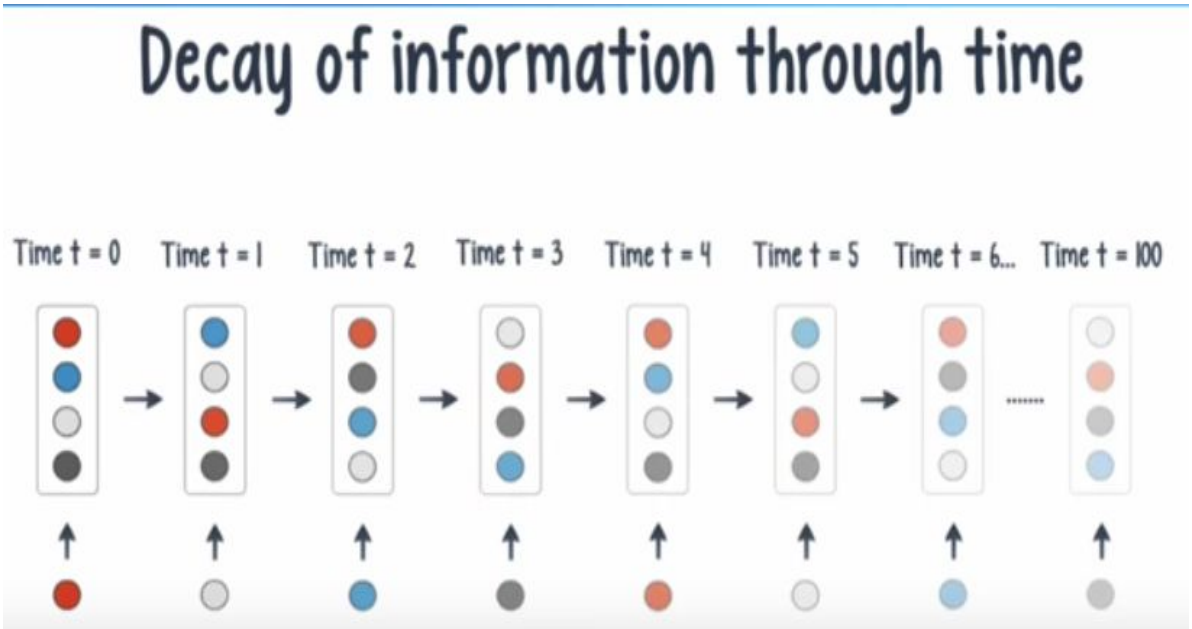
So Recurrent neural nets can connect the previous data with the present task. But there is problem in using RNN, Consider an example

“ I hope Senpai will notice me. Like walking through a barren street in a crumbling ghost town, isolation can feel melancholy and hopeless.Yet, all it takes is an ordinary flower bud admist the desolation to show life really can exist anywhere. This is similar to Stephen’s journey in The Samurai’s Garden.This novel is about an ailing Chinese boy named Stephen who goes tp Japanese village during a time of war between Japan and China to recover from his disease. She is my friend. He is my Senpai.””

Let’s say we want to train a model predict the last word Senpai given all previous words.

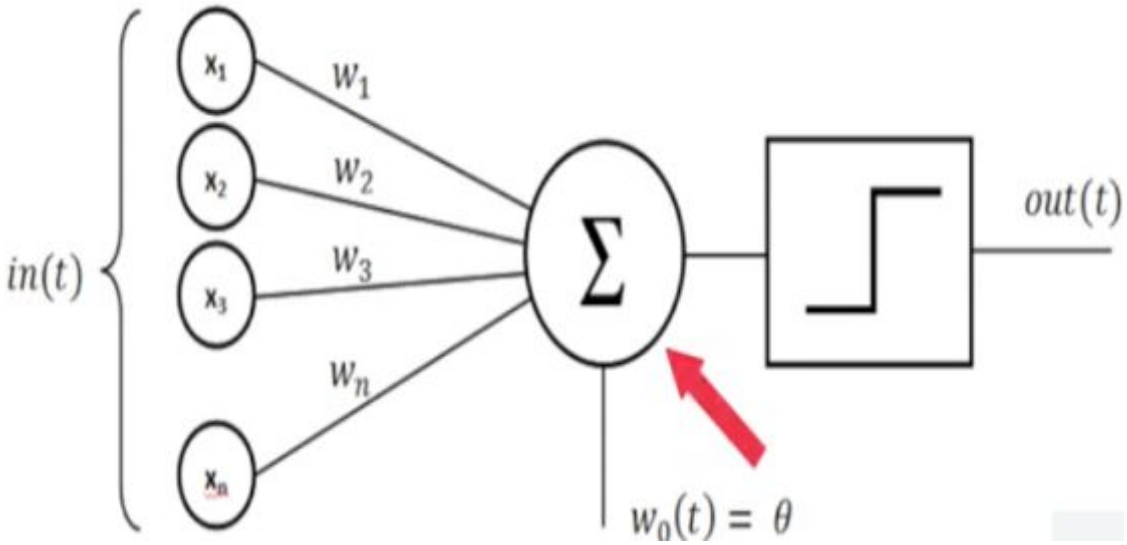
We need the context from the very beginning of the sequence to know that this word is probably Senpai.

In a regular neural net memories become more subtle as they fade into the past since the error signal from the later time steps doesn't make it far enough back in time to influence the network at early time steps during the back propagation. This is what we call vanishing gradient problem.

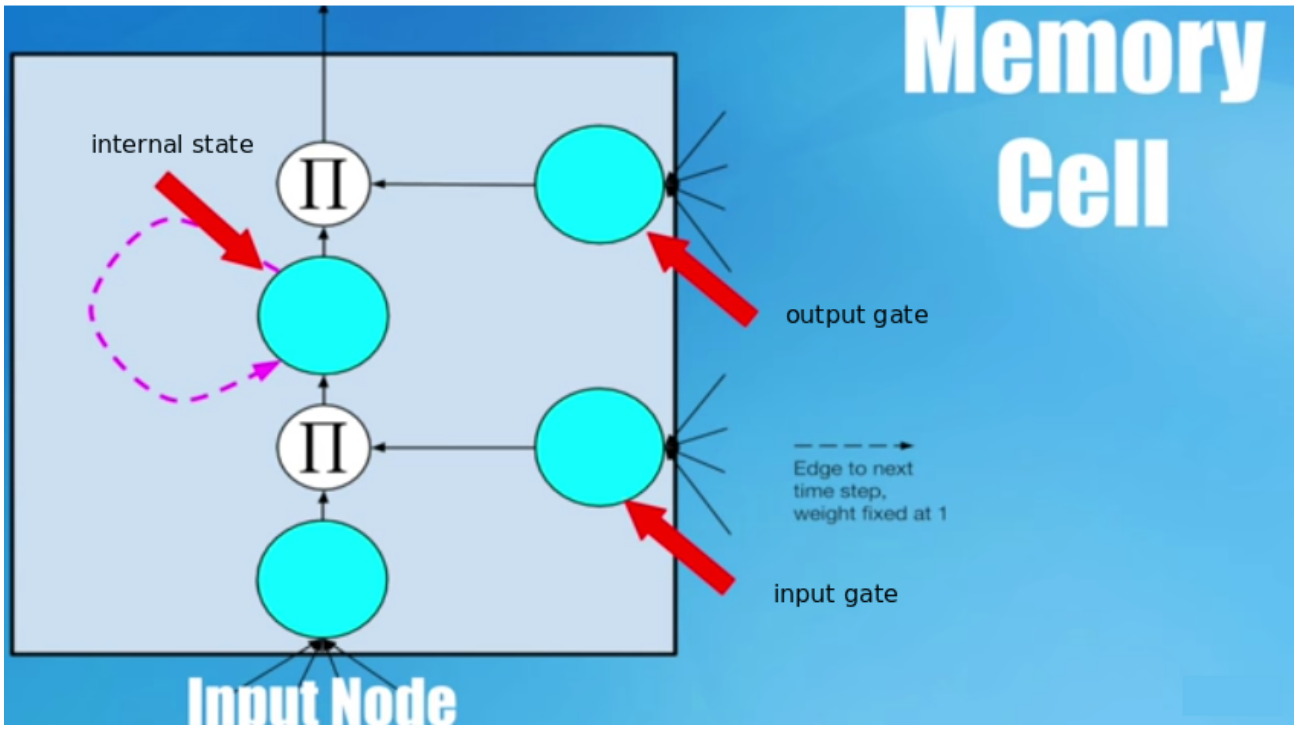


Vanishing gradient problem by Yoshua Bengio:
http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf

A popular solution to this a modification to recurrent neural nets called Long Short Term Memory (LSTM).Normally neurons are units that apply an activation function, like a sigmoid to a linear combination of their inputs.



In an LSTM recurrent net, we instead replace these neurons with memory cells.
Each cell has an input gate , an output gate and an internal state that feeds into itself across time step with a constant weight of 1. This eliminates the vanishing gradient problem since any gradient that flows into this self recurring unit during back propagation is preserved indefinitely, So this helps RNN to remember long term dependencies.



Ok now that we are clear why we add LSTM in code, now we can proceed further,

Code :
#adding a drop out of 20%
model.add.Dropout(0.2)

Similar to the previous LSTM we'll now add one more LSTM
The second LSTM layer outputs a prediction vector instead of a prediction value as a ouput

3. Now we'll use the linear dense layer to aggregate the data from the previous LSTM layer prediction into one single value

4.Then I'll compile my model using a popular loss function like mean squared error and gradient descent as a optimizer

5.Final I'll test the model on test data to check its performance.

Evaluation metrics:

To track and reduce the loss function I'm using mean square as a metric and Gradient descent as an optimizer to reduce the cost function.Here both cost function and error metric both is MSE itself.