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.This is basically a regression problem and the reason for using LSTM along with Recurrent neural network is that it should be able to remember a sequence rather than just the preceding the value.
Mean squared error is used as both classification error and error metric. Now let's see why we have to use MSE instead of absolute mean squared error .

Mean Absolute error(MAE) and Root mean squared error are the most common metrics used to measure the accuracy for a continuous variable.Let's define these metrics

Mean Absolute Error: MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weights.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

If the absolute value is not taken(the signs of the errors are not removed), the average error becomes the Mean Bias Error(MBE) and is usually intended to measure average model bias.MBE can convey useful information, but should be interpreted cautiously because positive and negative errors will cancel out.

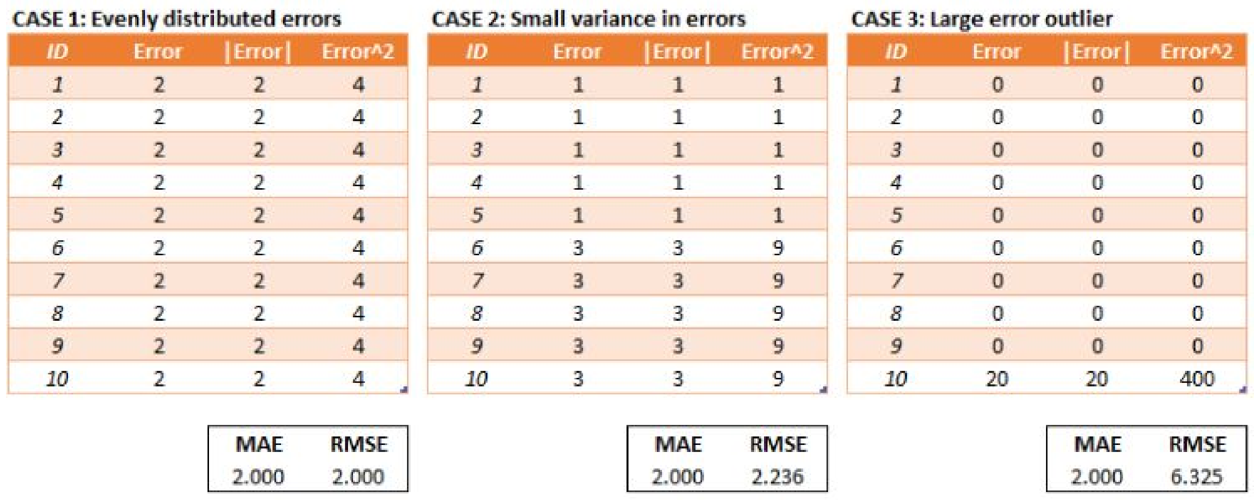
Root mean squared error(RMSE): RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It’s the square root of the average differences between prediction and actual observation

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Comparison

Similarities: Both MAE and RMSE express average model prediction error in units of the variable of interest. Both metrics can range from 0 to ∞ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better.

Differences: Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. The three tables below show examples where MAE is steady and RMSE increases as the variance associated with the frequency distribution of error magnitudes also increases.



CASE 1: Evenly distributed errors

ID	Error	|Error|	Error^2
1	2	2	4
2	2	2	4
3	2	2	4
4	2	2	4
5	2	2	4
6	2	2	4
7	2	2	4
8	2	2	4
9	2	2	4
10	2	2	4

MAE	RMSE
2.000	2.000

CASE 2: Small variance in errors

ID	Error	|Error|	Error^2
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	3	3	9
7	3	3	9
8	3	3	9
9	3	3	9
10	3	3	9

MAE	RMSE
2.000	2.236

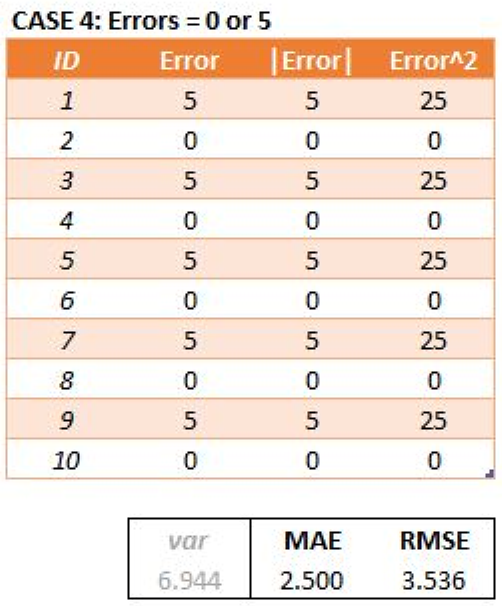
CASE 3: Large error outlier

ID	Error	|Error|	Error^2
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	20	20	400

MAE	RMSE
2.000	6.325

The last sentence is a little bit of a mouthful but I think is often incorrectly interpreted and important to highlight.RMSE does not necessarily increase with the variance of the errors. RMSE increases with the variance of the frequency distribution of error magnitudes.

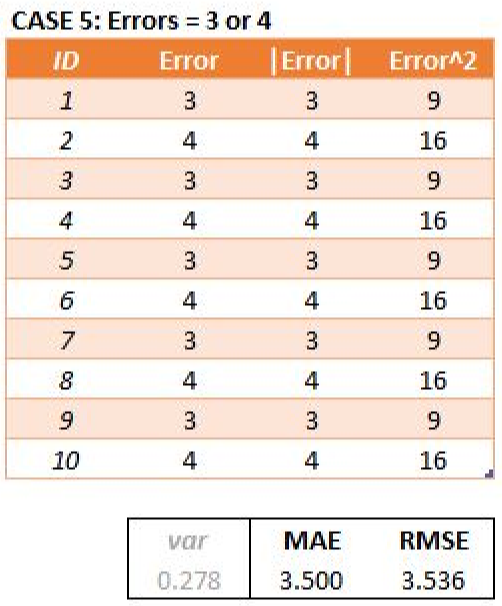
To demonstrate, consider Case 4 and Case 5 in the tables below. Case 4 has an equal number of test errors of 0 and 5 and Case 5 has an equal number of test errors of 3 and 4. The variance of the errors is greater in Case 4 but the RMSE is the same for Case 4 and Case 5.



CASE 4: Errors = 0 or 5

ID	Error	|Error|	Error^2
1	5	5	25
2	0	0	0
3	5	5	25
4	0	0	0
5	5	5	25
6	0	0	0
7	5	5	25
8	0	0	0
9	5	5	25
10	0	0	0

var	MAE	RMSE
6.944	2.500	3.536



CASE 5: Errors = 3 or 4

ID	Error	|Error|	Error^2
1	3	3	9
2	4	4	16
3	3	3	9
4	4	4	16
5	3	3	9
6	4	4	16
7	3	3	9
8	4	4	16
9	3	3	9
10	4	4	16

var	MAE	RMSE
0.278	3.500	3.536

There may be cases where the variance of the frequency distribution of error magnitudes (still a mouthful) is of interest but in most cases (that I can think of) the variance of the errors is of more interest.

Another implication of the RMSE formula that is not often discussed has to do with sample size. Using MAE, we can put a lower and upper bound on RMSE.

1. [MAE] ≤ [RMSE]. The RMSE result will always be larger or equal to the MAE. If all of the errors have the same magnitude, then RMSE=MAE.

2. [RMSE] ≤ [MAE * sqrt(n)], where n is the number of test samples. The difference between RMSE and MAE is greatest when all of the prediction error comes from a single test sample. The squared error then equals to [MAE^2 * n] for that single test sample and 0 for all other samples. Taking the square root, RMSE then equals to [MAE * sqrt(n)].

Focusing on the upper bound, this means that RMSE has a tendency to be increasingly larger than MAE as the test sample size increases.This can problematic when comparing RMSE results calculated on different sized test samples, which is frequently the case in real world modeling.

Conclusion

RMSE has the benefit of penalizing large errors more so can be more appropriate in some cases, for example, if being off by 10 is more than twice as bad as being off by 5. But if being off by 10 is just twice as bad as being off by 5, then MAE is more appropriate.

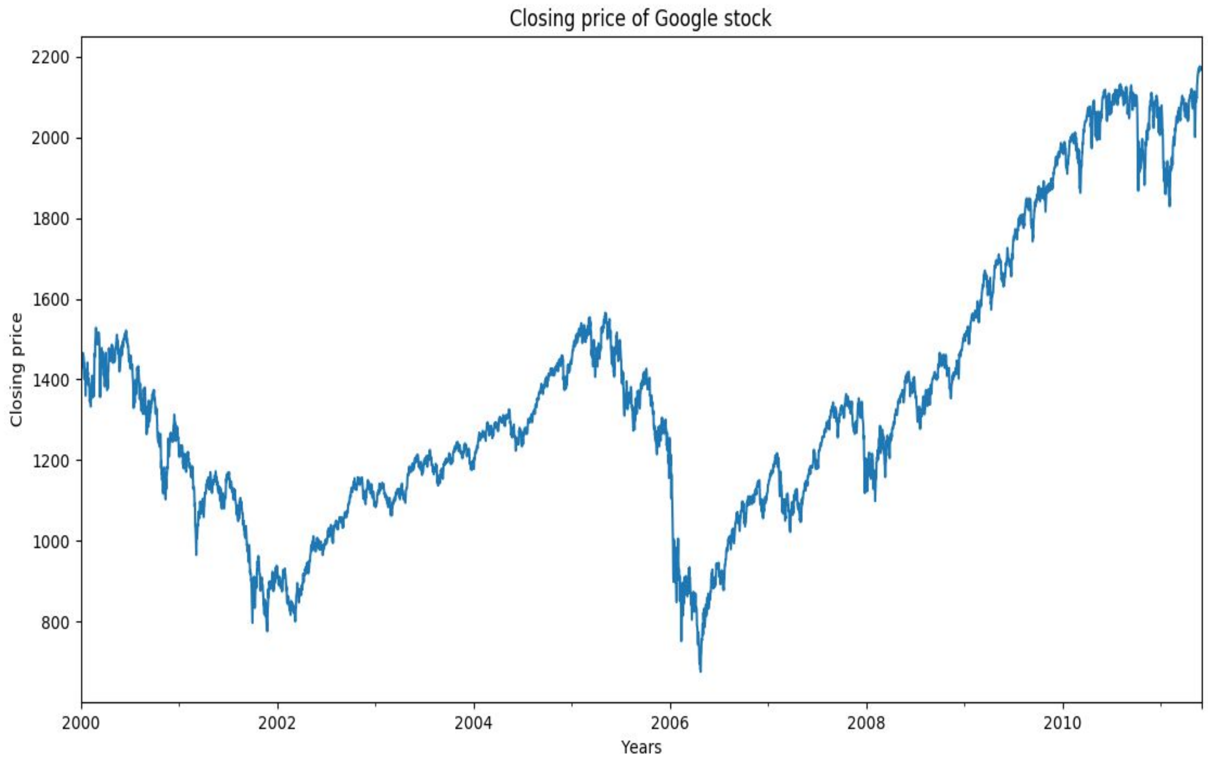
From an interpretation standpoint, MAE is clearly the winner. RMSE does not describe average error alone and has other implications that are more difficult to tease out and understand.

On the other hand, one distinct advantage of RMSE over MAE is that RMSE avoids the use of taking the absolute value, which is undesirable in many mathematical calculations

And the main reason behind using RMSE over MSE is that RMSE returns values in their standard units.

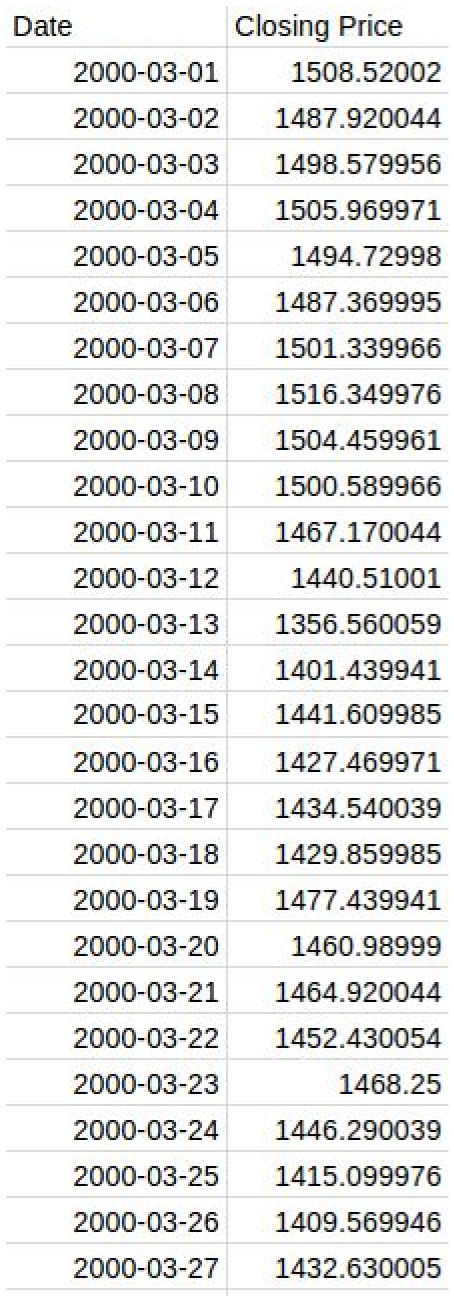
Data sets and inputs:

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/0B26anC5sNYXqTUxuTVlSaFIrSVE/view?usp=sharing

Let’s take a look at the Google closing price in the month of April 2000



Date	Closing Price
2000-03-01	1508.52002
2000-03-02	1487.920044
2000-03-03	1498.579956
2000-03-04	1505.969971
2000-03-05	1494.72998
2000-03-06	1487.369995
2000-03-07	1501.339966
2000-03-08	1516.349976
2000-03-09	1504.459961
2000-03-10	1500.589966
2000-03-11	1467.170044
2000-03-12	1440.51001
2000-03-13	1356.560059
2000-03-14	1401.439941
2000-03-15	1441.609985
2000-03-16	1427.469971
2000-03-17	1434.540039
2000-03-18	1429.859985
2000-03-19	1477.439941
2000-03-20	1460.98999
2000-03-21	1464.920044
2000-03-22	1452.430054
2000-03-23	1468.25
2000-03-24	1446.290039
2000-03-25	1415.099976
2000-03-26	1409.569946
2000-03-27	1432.630005

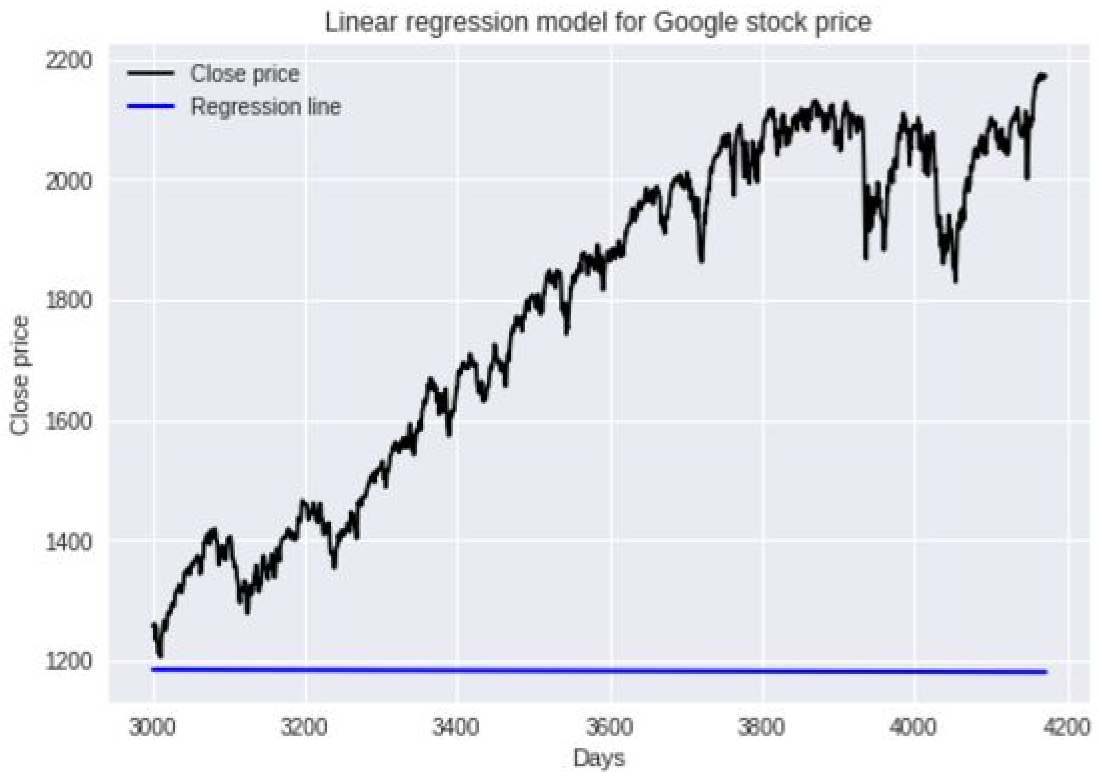
What are the maximum,minimum closing prices and other statistics of dataset?

Maximum price: 2175.030029 units on May 27 2011
Minimum price: 676.530029 units on April 26 2006
Mean price: 1355.746317207862 units
Standard deviation : 342.44313224454845 units

Abnormalities within the data: As we can see from the Closing price of Google stock figure that there few abnormalities within the data.For example consider the lowest price point on April 26 2006.

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

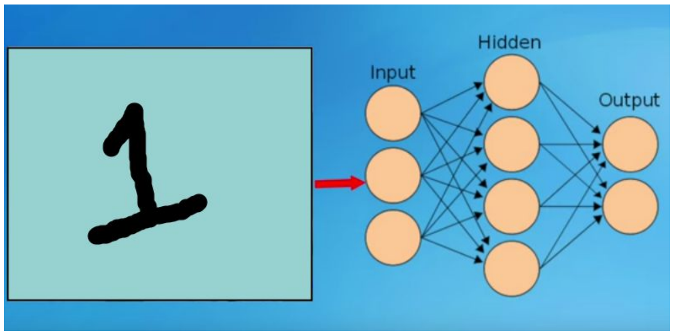
Workflow of project:

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:

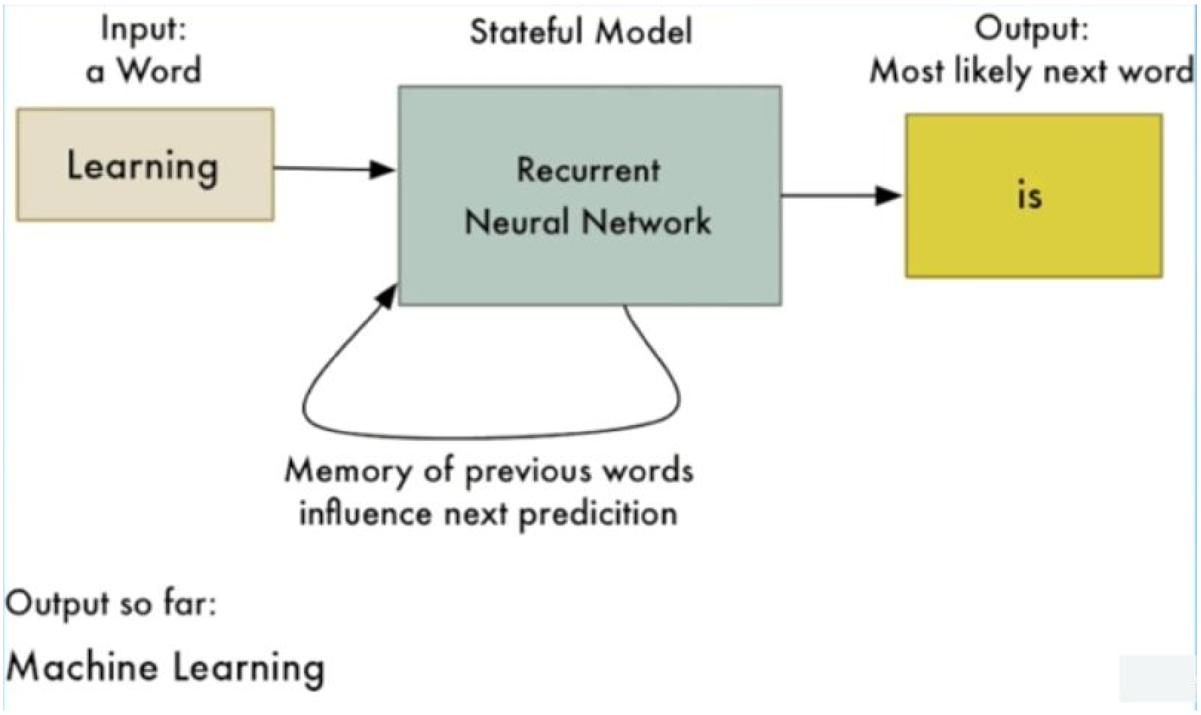
```
Model = Sequential()

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.

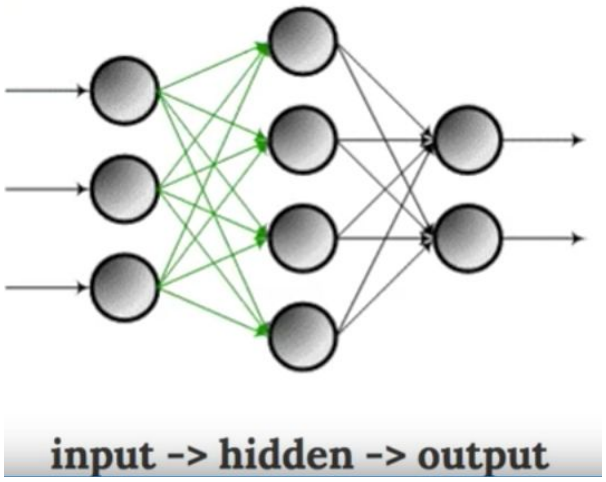


So we couldn't use it to predict the next frame in a movie because that would require a sequence of image vectors as inputs. And not just one since the probability of a certain event happening would depend on what happened every frame before it.

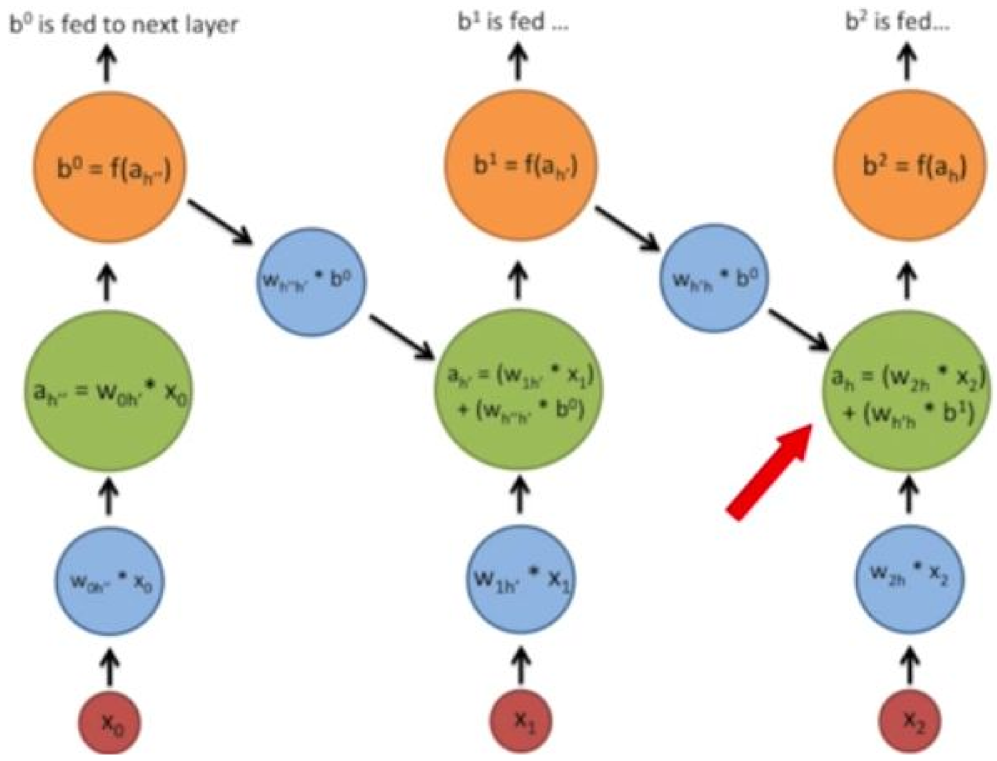
So we need a way to allow information to persist and that's why we can use a recurrent neural net. Recurrent neural nets can accept sequences of vectors as inputs.



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



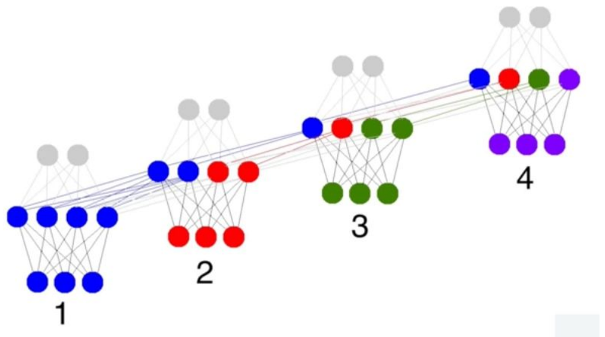
But in a recurrent net, the hidden layer is a combination of the input data at the current time step and the hidden layer at a previous time step. The hidden layer is constantly changing as it gets more inputs and the only way to reach these hidden states is with correct sequence inputs. This is how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



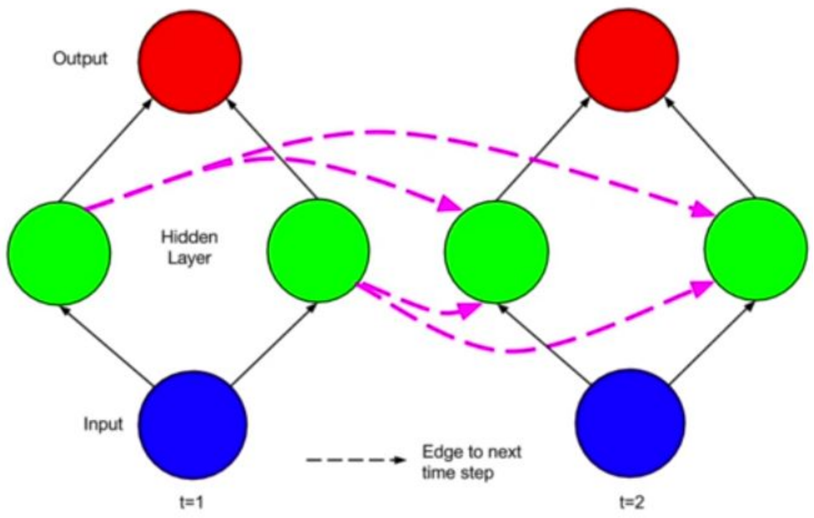
ht = ϕ (W * Xt + U * h(t-1))

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.



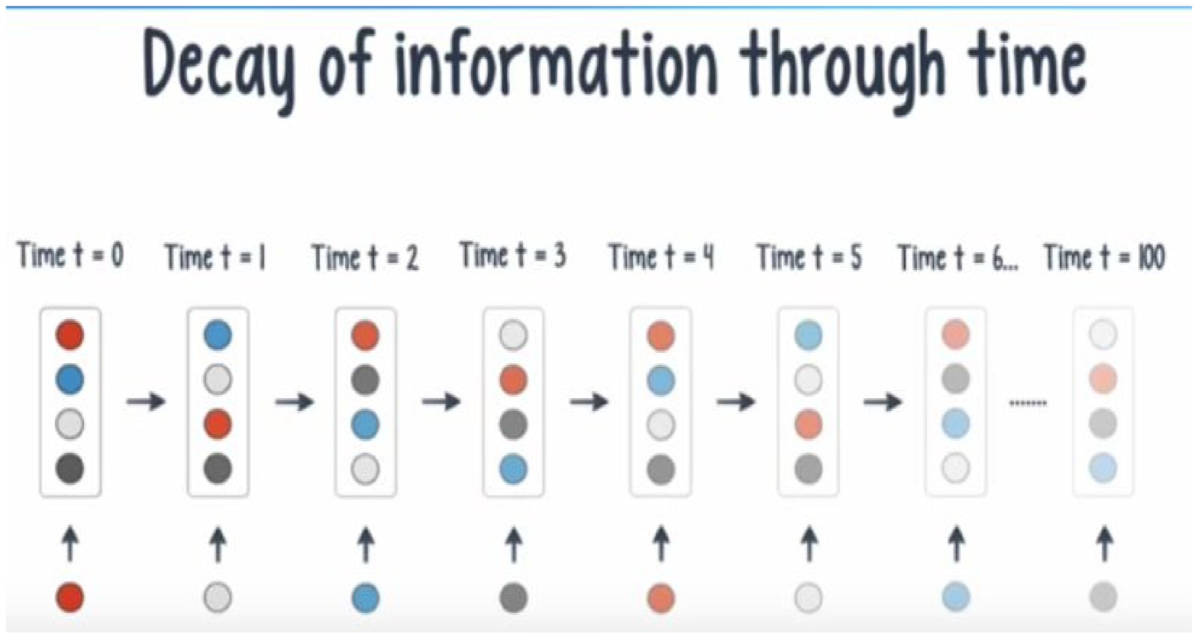
So Recurrent neural nets can connect the previous data with the present task. But there is problem in using RNN, Consider an example

“ 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.

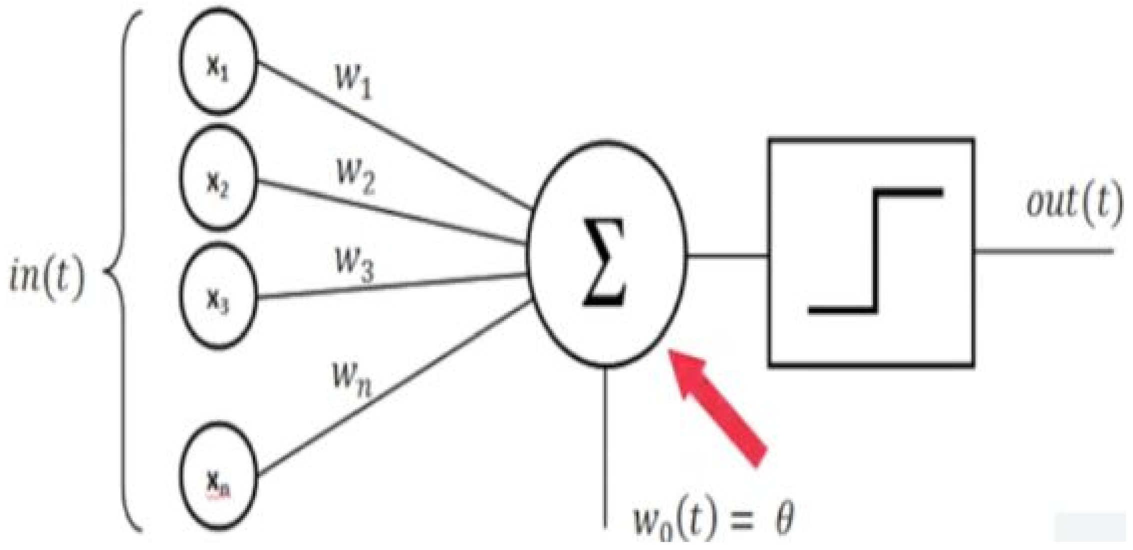
We need the context from the very beginning of the sequence to know that this word is probably Senpai.

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.

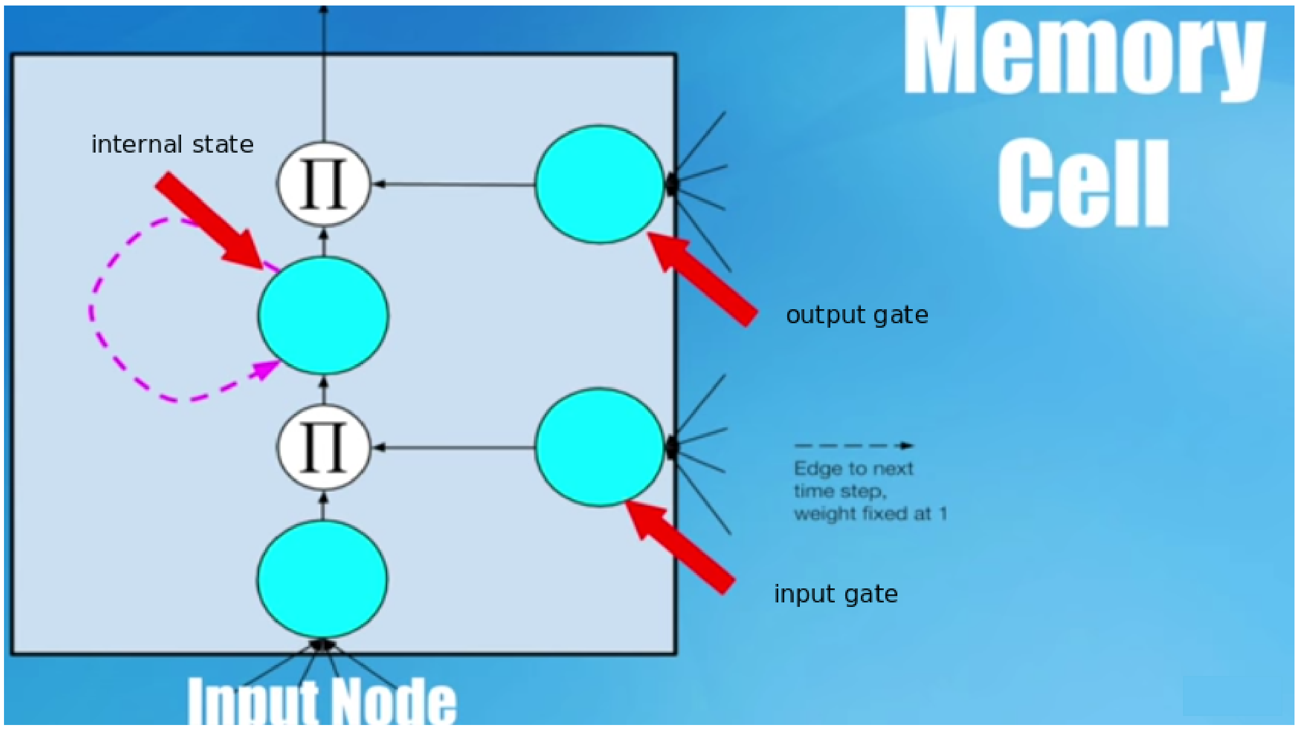


Vanishing gradient problem by Yoshua Bengio:
http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf

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)
```

Now let’s see what is Dropout and why we should use it?
Dropout is a simple and powerful regularization technique for neural networks and deep learning models.

Dropout is a regularization technique for neural network models proposed by Srivastava, et al. in their 2014 paper Dropout: A Simple Way to Prevent Neural Networks from Overfitting Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

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.