

## Stock Price predictor with Recurrent neural nets

### PROJECT OVERVIEW

The transition from traditional auction in 1970s to computerized transactions was fuelled with a need for an efficient access to the market. This set a stage for algorithmic and high-frequency trading. An estimated 70% of US equities in 2013, accounted for by automated trading [1]. Investment firms and hedge funds have used mathematical modelling to predict market trends. They aim towards maximizing the return and spreading their risk over different financial assets. These approaches can be classified as Follow-the-Winner, Follow-the-Loose', Pattern-Matching and Meta-learning. Though these traditional methods are efficient, their performance is dependent on the validity in different types of markets.

Quantitative strategies of hedge funds have received considerable returns over decades. Application of growing computing power and availability of big data has allowed models to identify and harvest on market inefficiencies [2]. These organizations are actively looking for AI solutions to outperform the benchmark indexes. In recent times a large ecosystem of start-ups has unfolded, centred around FinTech, with the aim of providing AI solutions for investment and saving.

The project aims towards utilizing Recurrent Neural Nets to predict the stock prices. The goal here is to leverage modelling abilities of neural networks for time series forecasting [3]. Time series forecasting is a difficult type of predictive modelling problem, as it has added complexity of sequence dependence among the input variables [7]. Recurrent neural networks have been quite successful in modelling time series data. LSTM is a type of RNN which can model short-term memory over a long period of time. LSTM enables us to model sequence dependence as short-term memory and incorporate it as a feature in our model.

## PROBLEM STATEMENT

The goal of this project is to predict future adjusted price for a given stock across a given period. With vast amount of historical data, application of machine learning techniques becomes appropriate here, to predict price trends for a stock being traded in a market. These models should be tested for accuracy and validated for performance. Because of the availability of wide range of historical data, a model of this type can be trained over a long period of time and can make predictions with a time series.

This project aims to answer these questions

- Can a deep learning model beat a machine learning benchmark model for stock price prediction?
- Which architecture/ hyperparameters is gives an optimal result for the model?
- Can a simple deep learning model perform good enough to drive the investment strategies?

## METRICS

It is important to measure the quality of a model by quantifying its performance. Here we are going to use Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). It is the difference between the price predicted by a model and actual price, a stock was trading on a day. Mathematical formula of RMSE is as depicted in the Fig 1. Using RMSE gives us consistency while comparing with other studies. Also, we will use graphical visualizations of different models to find degree of fit, ability to incorporate volatility and lag across time and other intrinsic micro trends from dataset.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

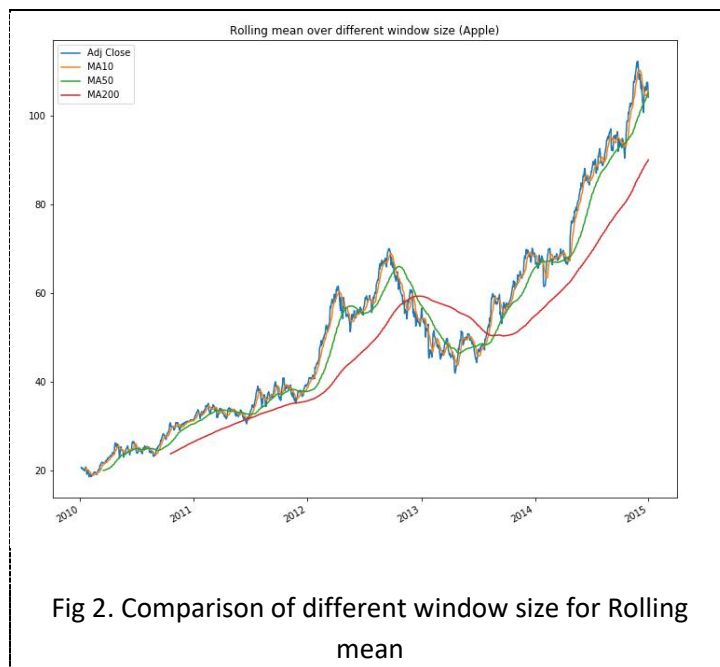
Fig 1. Root Mean Square Error (RMSE)

## ANALYSIS

### DATA EXPLORATION & EXPLORATORY VISUALIZATION

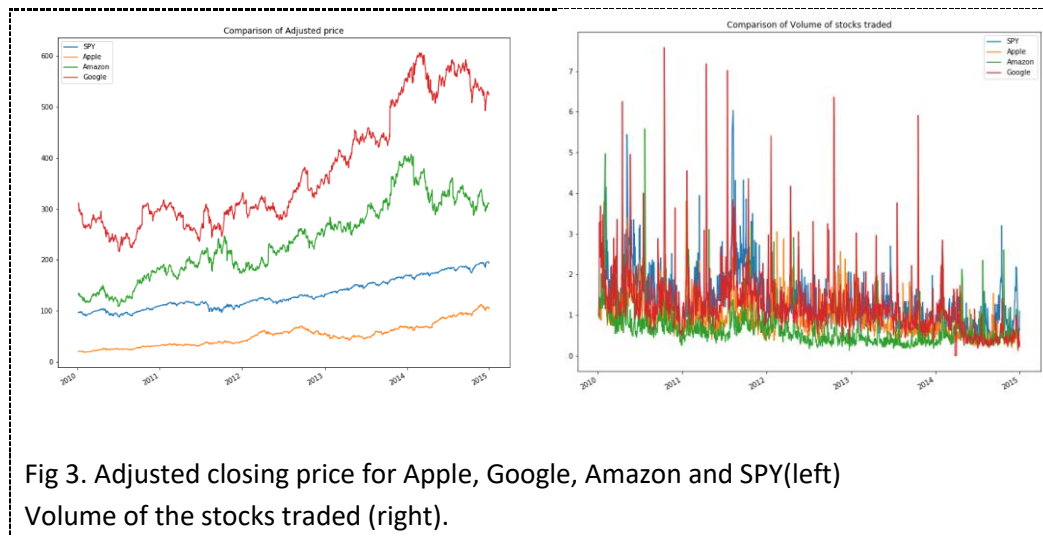
In this project we are going to analyse stocks of Apple (AAPL), Google (GOOG), Amazon (AMZN) and SPY stock indices. The historical data for these entities was retrieved from Yahoo finance application and is available in repository inside 'Project/data' folder. A separate notebook for each entity having detailed analysis can be found inside the 'Project/notebooks' folder. All the images of the report can be found in 'Project/images' folder.

Here, we are using pandas to load a csv file for a given stock (in our case 'data/AAPL.csv'). Date column is parsed while loading and set as an index. As we are going to use 'Rolling mean and Adjusted closing price as a feature for our model, all the columns except this were dropped. For rolling mean, window size of 10 is selected. It is evident from the Fig 2 that resolution of window size 10 is better than the other window size.

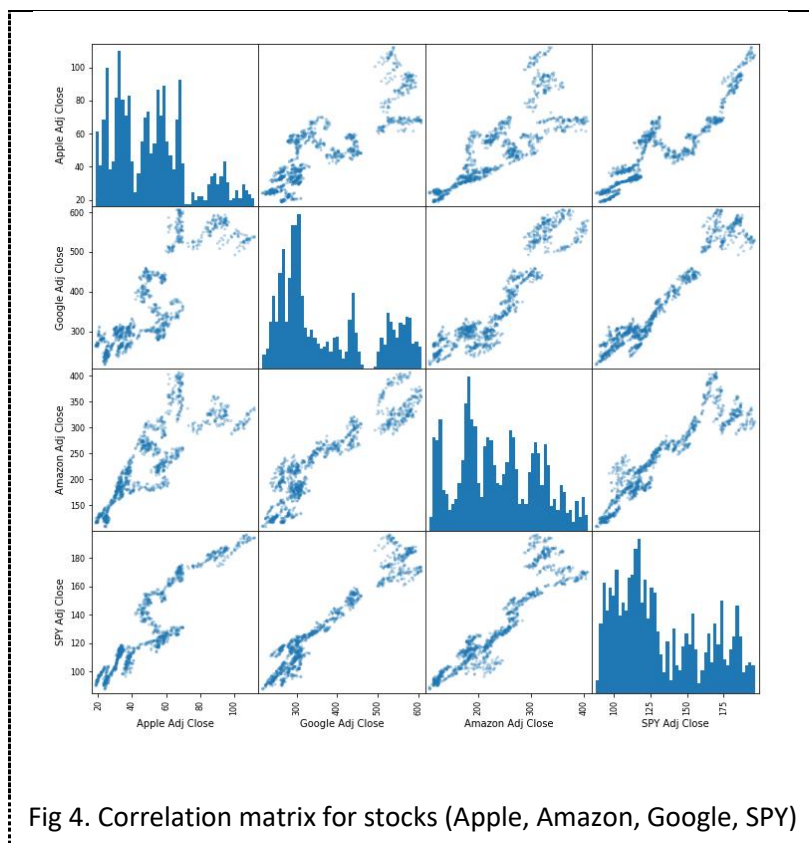


We are using past five-year data to create and validate models. Data ranges from Jan 2010 to Jan 2015. Fig. 3 depicts Adjusted closing price for stocks in these date range. For Adjusted closing price it looks like google has always been much more valuable as a company. But to better understand this we must look at total market cap of the company not just the stock prices. This information though absent, can be derived from total value traded for a given

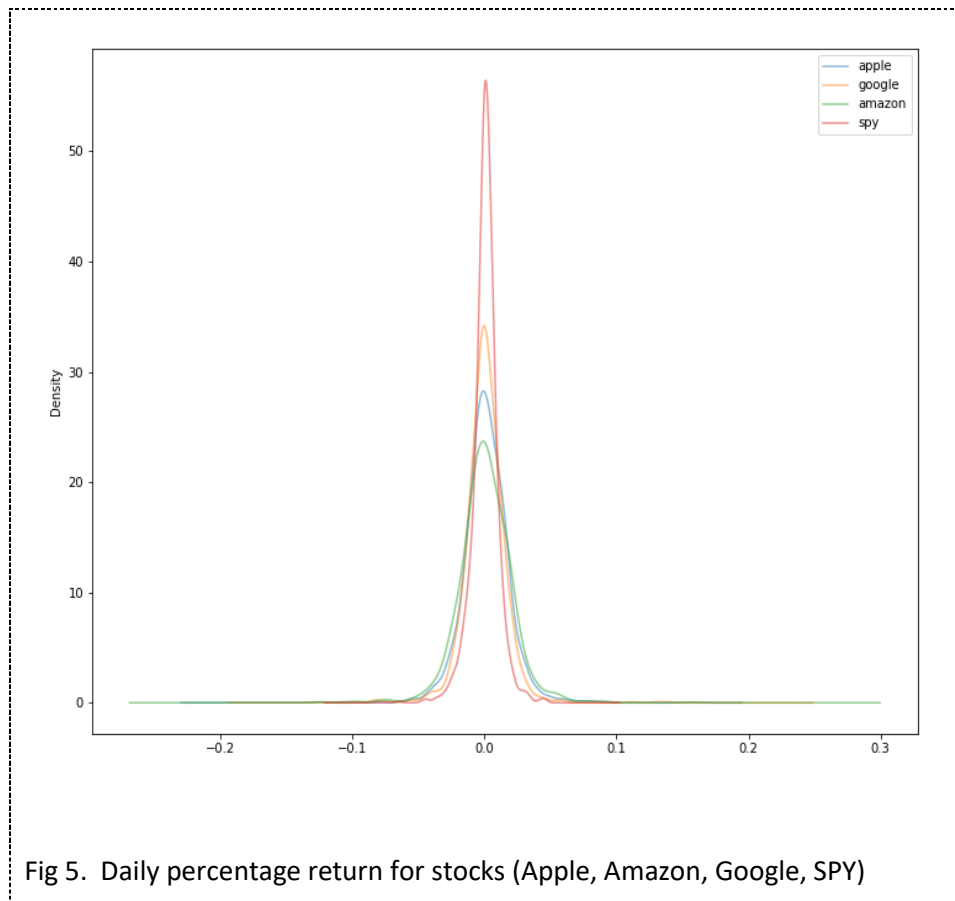
period. In Fig 3, we can also see volume trade plots for different stock, using both values, market cap of a company can be approximated.



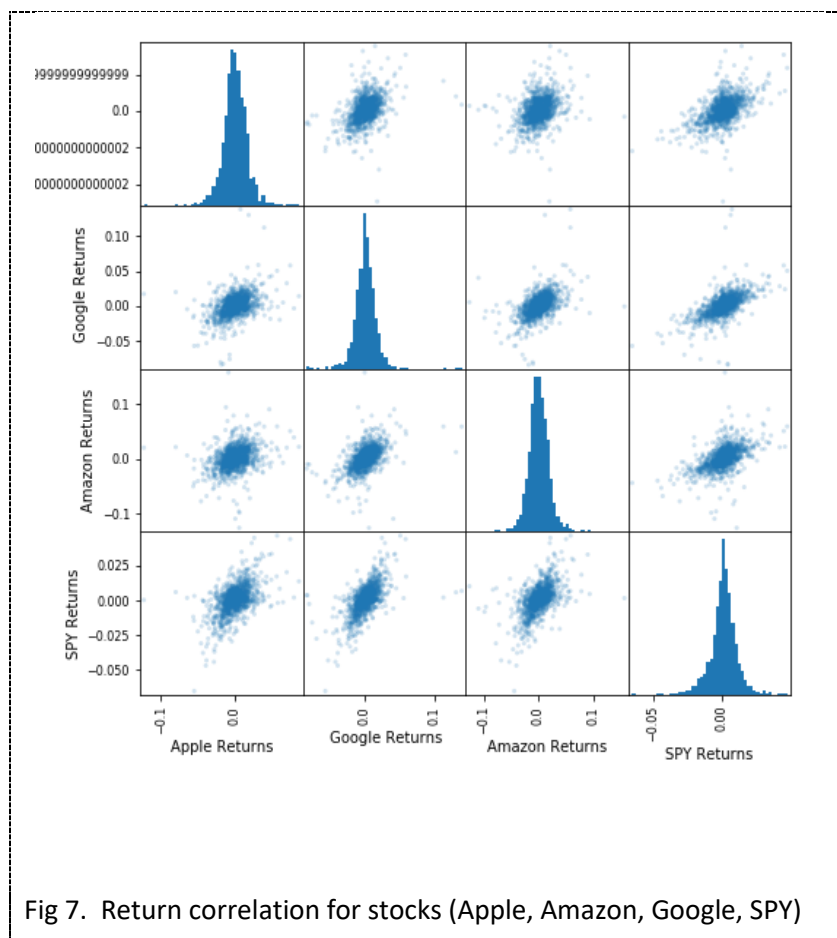
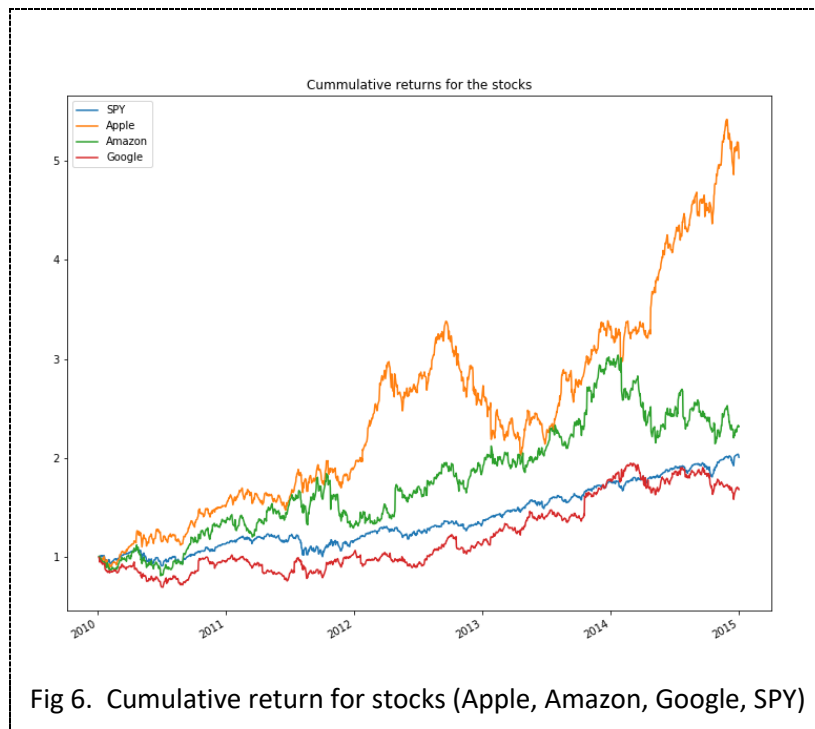
These entities share a common domain, i.e. software industry. This correlation can be explored. Correlated entities react in a similar fashion to external events to the industry they belong. For example, local government policies, effect of season and climates etc. In Fig4, we can observe that these is a certain degree of correlation between these stocks.



Daily percentage return is a good metric to estimate daily volatility of a given stock. It basically indicates, if a stock was bought today and sold tomorrow, how much gain or loss it will make. Fig5 plots the daily percentage returns of stocks. It can be observed, all the stocks we are working with, has a similar degree of volatility.



While daily returns are useful, it doesn't give the investors an immediate insight to the gains they have made till date, especially in cases when stocks are very volatile. Cumulative returns are computed relative to day investment was made. If cumulative returns are above 1, it is making profit and vice versa. Let's explore cumulative returns. In Fig6, it can be observed that Apple gave a good cumulative return compared to others. In Fig7, correlations between returns of different stocks can be observed.



## ALGORITHMS AND TECHNIQUE

### Algorithm

The problem statement of this project revolves around making predictions for a given time series of stock prices. The project includes studies utilizing Basic LSTM, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. Standard RNNs suffers from vanishing and exploding gradient problems. Because of which, learning long term temporal dependencies are difficult in them. LSTMs and GRUs are types of RNN. They use special units (and gates) in addition to standard units. A set of gates in them, allow to control when an information is remembered, when it is forgotten and when it is passed to output. This architecture allows them to learn long term dependencies. GRUs are like LSTMs but uses simplified structures. They use gates to control the flow of information but uses fewer gates and no separate memory cells.

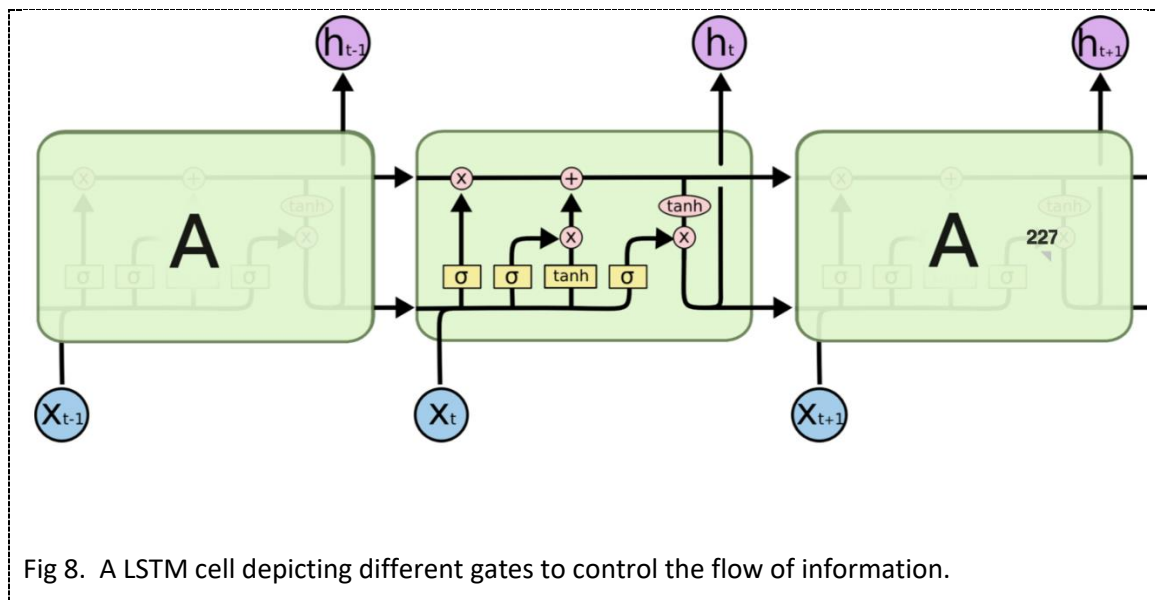
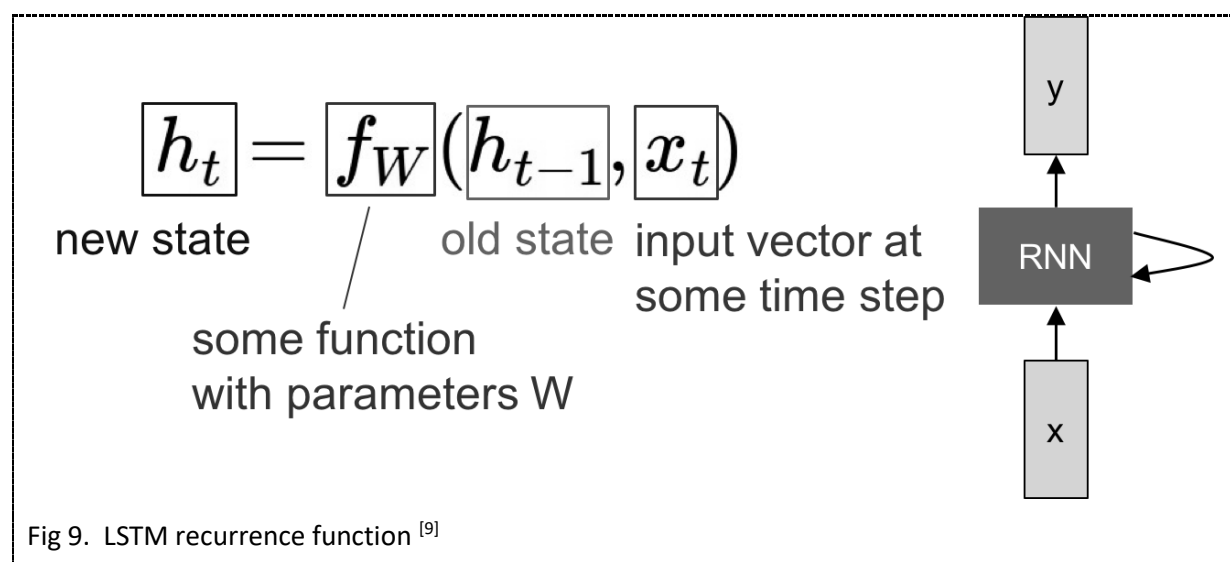


Fig 8. A LSTM cell depicting different gates to control the flow of information.

The project includes experiments with these three types of RNNs, with a goal for increasing the prediction accuracies of stock price.

### Algorithm Description and Justification

A normal feed forward networks can learn a relationship between inputs and outputs. They have limitations when ordering of data matters, and output at a given time step depends on the events happened in previous time steps, e.g. stock prices. In such cases, a network needs to incorporate information from previous event to give a sensible prediction. RNNs outshines here as it maintains a hidden state which is passed from previous time step, back into the network at the next time step. Hidden recurrence learns what to remember, what to forget and what to pass to the network. In this case input to the network is hidden state from previous time step along with the input. The flow of this values is controlled through different gates, parameters of which are learned during training.



From above formula it can be observed that current hidden state  $h(t)$  is a function of previous hidden state  $h(t-1)$  and the current input  $x(t)$ . The network learns to use  $h(t)$  as a lossy summary of the task-relevant aspects of the past events. Total loss for a given window/sequence of inputs paired with sequence of targets will be the sum over all time steps.

The RNNs are promising with respect to learning temporal dependencies in input data. The available context in hidden states may allow RNNs to learn trend (upwards, downwards) and seasonality (repeating patterns over time) in data. Traditional time series methods use



univariate data with linear relationships and fixed and manually-diagnosed temporal dependence. RNNs add the explicit handling of ordered observations and the promise of learning temporal dependence from hidden state context. This makes RNNs suitable for the problem statement of this project <sup>[10]</sup>.

### *Algorithm Parameters*

In addition to optimize model architecture, following set of parameters can be tuned to improve the accuracy and reliability of the model predictions. In project, these parameters are encapsulated in a class, instance of which is used while training and testing. It can be found at 'Project/deepstocks/common\_configs.py'.

- Input parameters
  - Pre-processed and normalized Adjusted closing price
  - Rolling mean (window=10) of Adjusted closing price
  - Keep probability
- Neural network architecture
  - Model type (LSTM, GRU, BasicLSTM)
  - Number of nodes or neurons per layer (tested 64, 100, 128, 156, 256, used: 156)
  - Number of layers (tested 1, 2, 3, used:2)
- Training parameters
  - Train/validation/test split (proportion of data for training, validation and testing, used 70%, 15 %, 15% respectively)
  - Batch size (how many time steps to include during single time step, tested: 50, 64, 128, used: 64)
  - Optimizer function (function to optimize and minimize the loss, used Adam optimizer throughout)
  - Epochs (how many times to train over the dataset, tested: 28, 56, 64, 150, 200, used: 64)
  - Learning rate

## Techniques

### 1. Loading the data

- A dataset of a given stock is loaded into a pandas dataframe for a given date range.
- Date values is set as index, and all the columns except 'Adj Close' and 'Volume' are dropped from dataframe.
- 'Adj Close' column is normalized to have values between 0 and 1.
- A column with name 'Rolling mean' is added to the dataframe, the values of which is set to rolling mean on 'Adj Close' values with a window of 10 (using pandas.Series.rolling).

### 2. Train-valid-test split

- For training the model, 'Adj Close' and 'Volume' features from the above dataframe is used.
- This sequence is split into 70:15:15 ratio, 70% for training and other two for validation and testing.

### 3. Tensorflow graph

- A graph of model is created using different tensorflow operations.
- Different summaries are written to logs, to visualize better with tensorboard.

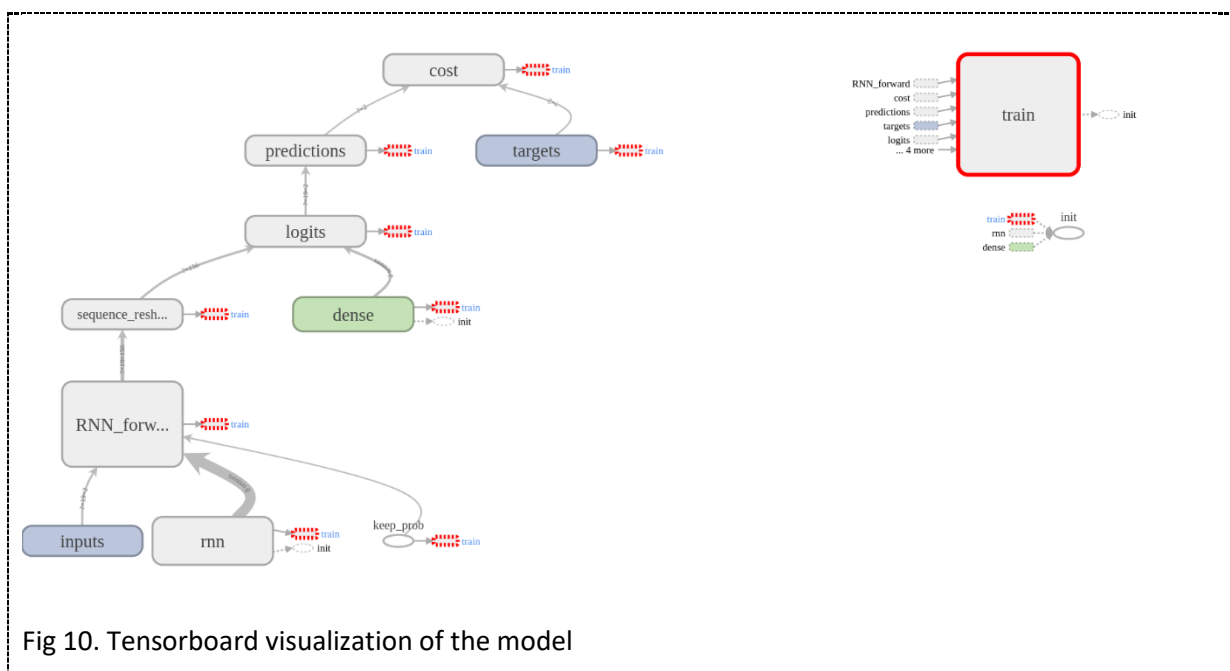


Fig 10. Tensorboard visualization of the model

## *BENCHMARK*

The benchmark for the given project is a linear regression model. The aim here is to build a model having better performance than this benchmark.

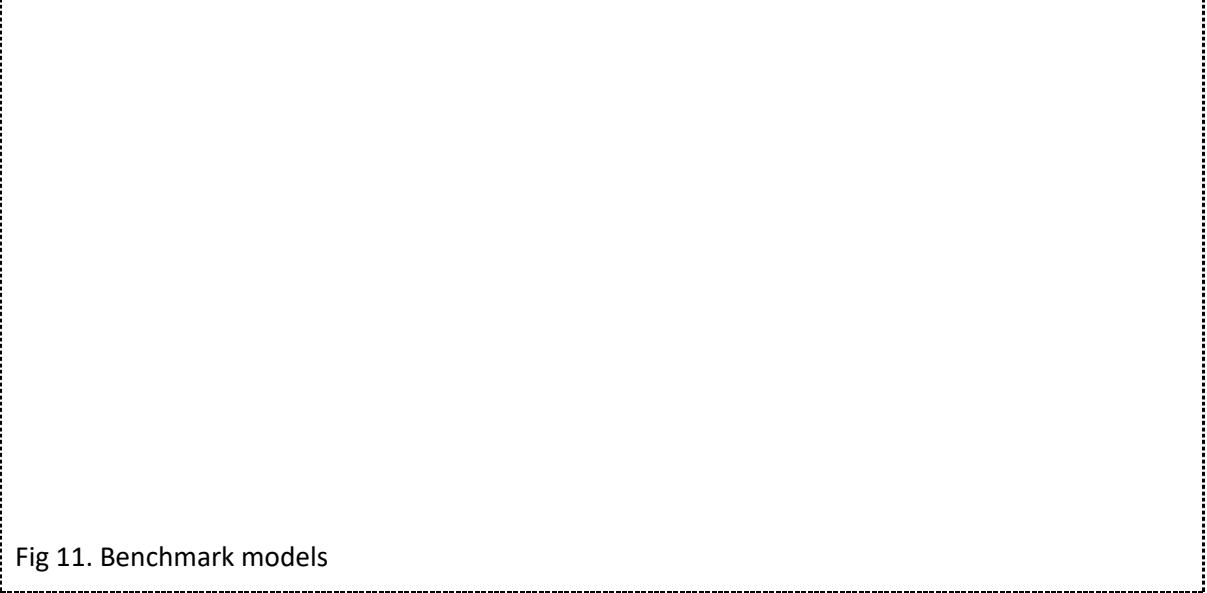


Fig 11. Benchmark models

## METHODOLOGY

### DATA PREPROCESSING

Acquisition and pre-processing of data is done as described in the sequence of steps below. To reuse across different notebooks and experiments, methods were modularized and can be used from 'Project/deepstocks/data\_loader.py' and 'Project/deepstocks/preprocess.py'.

1. Recently, Yahoo! has deprecated their Api interface for financial data. The data for this project was acquired using their web interface. Stock data for Apple, Amazon, Google and SPY were downloaded and stored inside 'Project/data' folder to reuse across all the experiments. Data is in csv format, having values ranging year 2009 to 2018 March.
2. Next step is to load the data from csv to pandas dataframe. Date column was parsed and made index in the dataframe. Columns loaded are 'Adj close', 'Volume', 'Date'.
3. A new column with the name 'Rolling mean' was added into the above dataframe. A window size of 10 is selected, considering the observations from Fig 2.
4. Values were normalized to the range [0.0, 1.0]. Sklearn MinMax scalar was used for this purpose.

Fig 12 gives an overview of lookback parameter. The value of lookback means how many previous days values a model should consider while making a prediction. Lookback = 1 means a model must consider only today's value to make an estimate. Similarly, a value of 5 will include prior four days also in input to the model. In this project Lookback = 20 is used, value of which is set in 'Project/deepstocks/common\_configs.py'. To prepare the dataset as shown in the Fig 12, method is defined at 'Project/deepstocks/preprocess.py' with name `prepare_data`. It expects a dataframe, lookback value, validation ratio and test ratio as input.

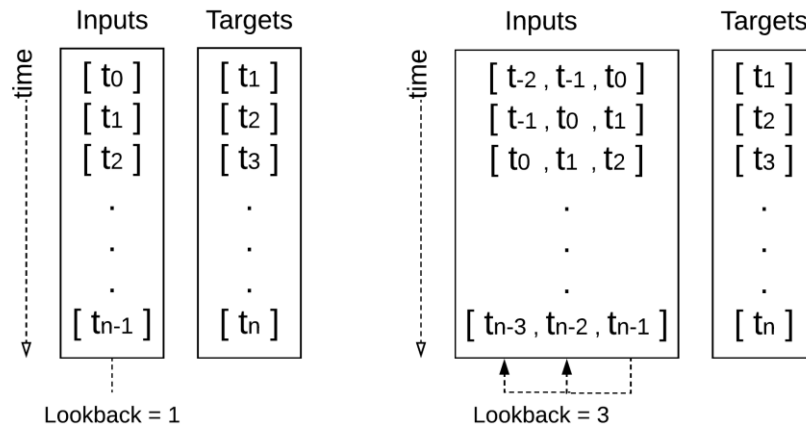


Fig 12. Dataset with lookback parameter 1 and 3. The value of lookback signifies the number of prior time steps included in each input to the model.

## IMPLEMENTATION

After a dataset is loaded and pre-processed as defined above, the implementation process across all experiments/ studies occurs as follow

- Define the training parameters (and hyperparameters). It is defined at common\_configs.py inside Project folder.
- Define model architecture (discussed later in this section)
- Compile the model
- Train and validate the model
- Save the model checkpoints
- Load the model and make predictions
- Calculate error
- Plot actual and predicted price
- Save notebook, graph and error values.

Broadly three types of model architectures were tested. The basic difference in them being the type of cell used, namely LSTMs, GRUs and BasicLSTM. Two hidden layers is used across all studies, as while training it was found that for layer greater than 2, model was overfitting

the training data. Keep probability 0.8 and learning rate 0.0008 is used (found to work well by hit and trials). In each hidden layer 156 nodes are used and leaky relu is used as an activation function. Adam optimizer is used in the model to minimize the loss value.

### REFINEMENT

The different studies in this project is iterative and aim towards finding a suitable model to maximize the performance. Study 1 examines the data through exploratory analysis. Study 2 and Study 3 sets the benchmark models. The remaining studies explore different model architecture of RNN.

- Study1 // Data exploration // Acquire and analyse stock data
- Study2 // Benchmark model // Linear Regression with scikit learn
- Study3 // Benchmark model // Polynomial regression with scikit learn
- Study4 // Deep Learning model // Stocks with BasicLSTMs
- Study5 // Deep Learning model // Stocks with stacked LSTMs
- Study6 // Deep Learning model // Stocks with GRUs

These studies can be found in Ipython notebooks, exported graph images in the github repository. For each variation of model architecture used, error rates (MSE) were logged and compared to previous iterations to check if changes are improvements towards a robust deep learning model.

The error rate highlighted in figure 14, indicate from model architecture perspective that either BasicLSTMs (study4) or LSTMs (study 5) displayed the best result. Other parameter (e.g. learning rate, epochs, batch size, number of layers, number of nodes) were estimated using hit and trial studies, notebooks of which are not included in the repository.

MSE	SPY	AAPL	AMZN	GOOG
BasicLSTMs (Study 4)	0.005630	0.002523	0.010148	0.009966
LSTMs (Study 5)	0.005596	0.002650	0.008268	0.014085
GRUs (Study 6)	0.006425	0.002644	0.009825	0.010366

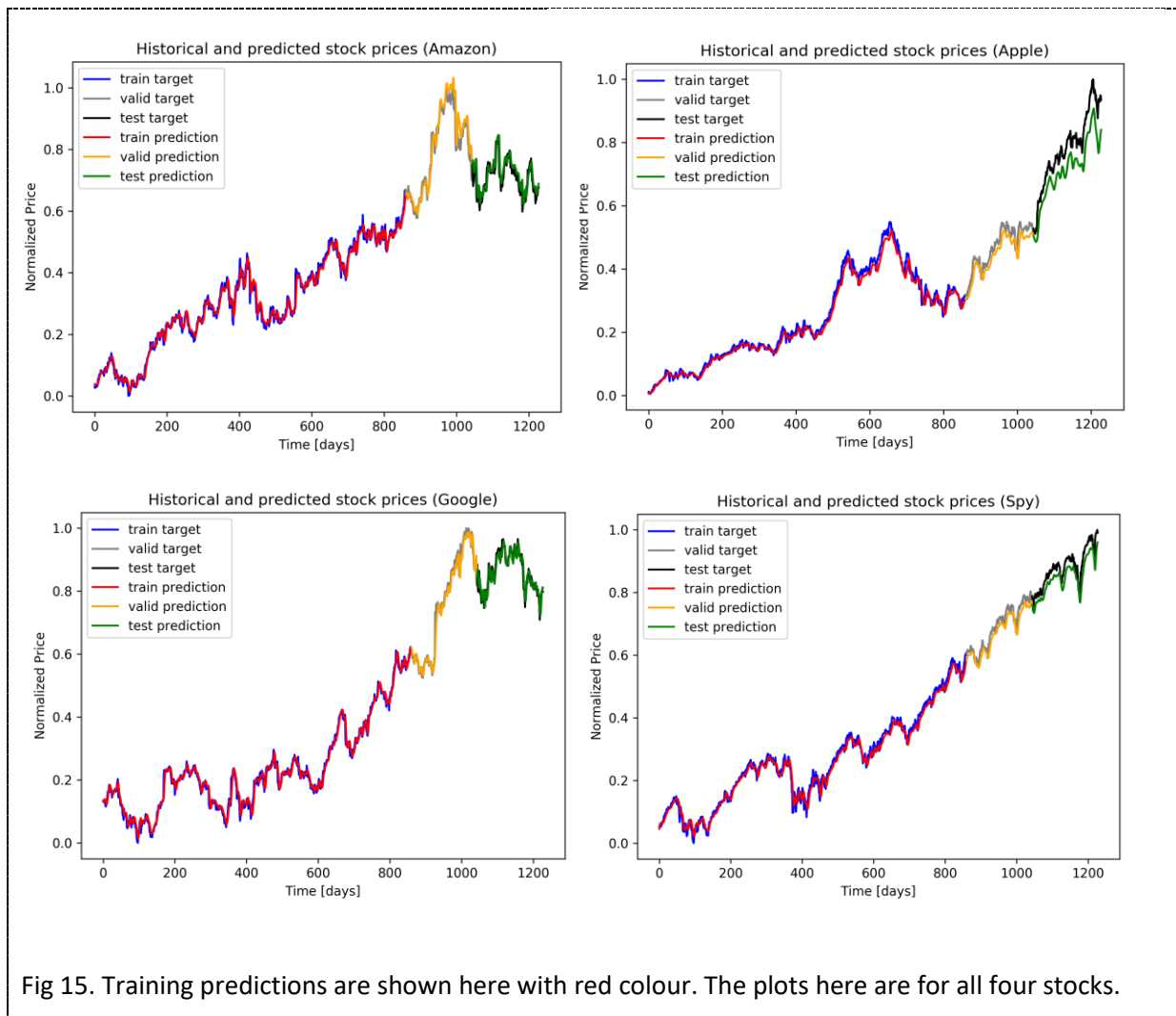
Fig 14. Loss values for models on validation dataset

## RESULTS

### MODEL EVALUATION AND VALIDATION

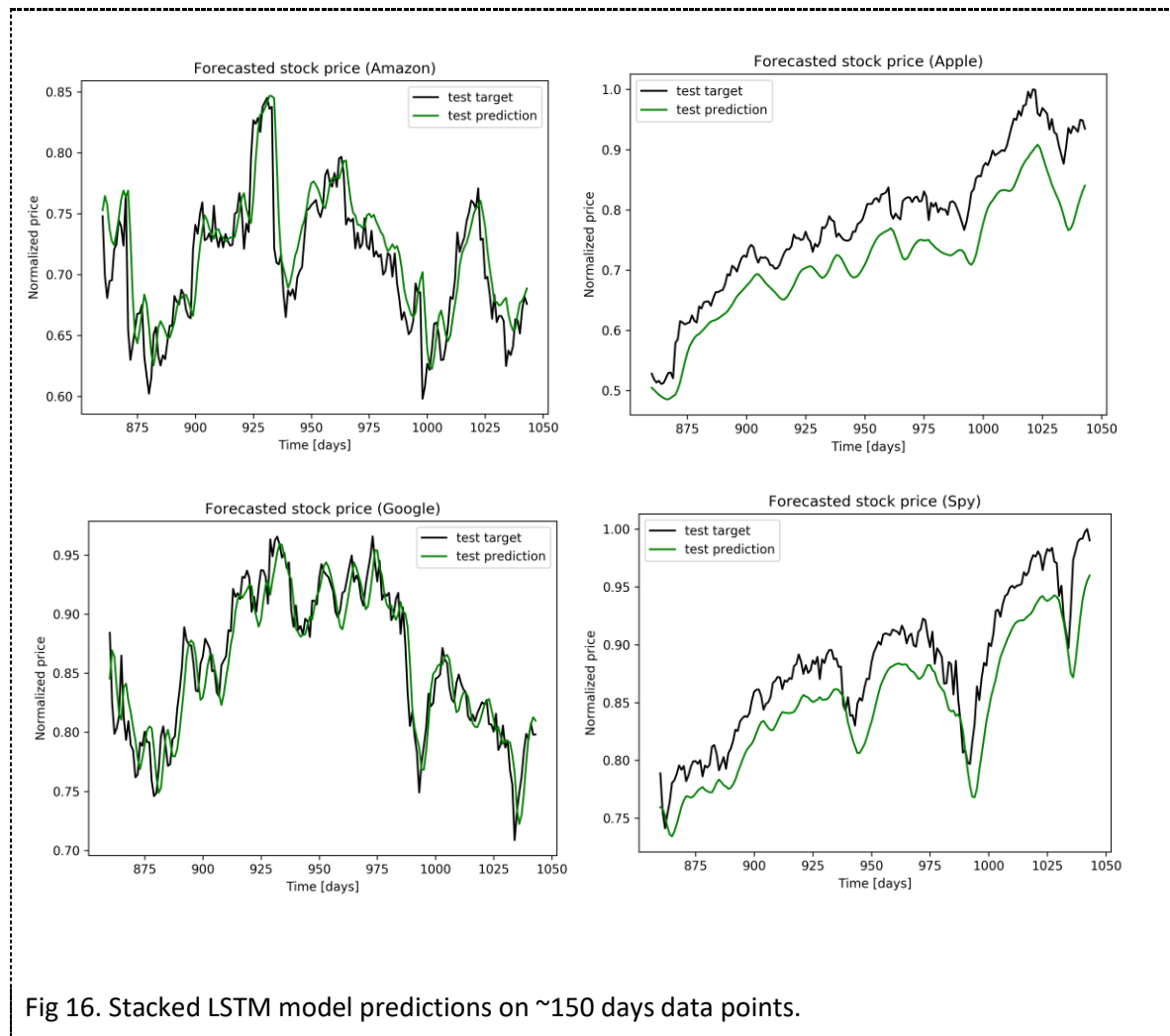
After comparing studies from previous section including architectures with BasicLSTM, LSTM and GRU cells, the two models which performed best across all four stocks were the Basic LSTM (study 4) and staked LSTM (study 5). Because the range of performance was narrower for study 5, for final study I selected the stacked LSTM with following model parameters.

- Learning rate: 0.0008
- Lookback: 20
- Batch size: 64



- Layers: 2
- Number of nodes (each layer): 156
- Epochs: 64
- Cell type: LSTM

A close fit in the predictions (fig 15) and the consistency of the final error metrics (fig 17) can be observed. Based on which it can be implied that this solution generalizes well across the four stocks. Also, based on the variability in the shape of closing price of test period and volatility in the data across more localized time frames, this model should be robust enough for the problem given a given stock has adequate size of dataset. It should be observed that error values are very less, and little compared to non-deep learning models.





	SPY	AAPL	AMZN	GOOG
MSE	0.005596	0.002650	0.008268	0.014085
Accuracy (%)	99.98581	99.94428	99.99018	99.98571

Fig 17. LSTM model metrics, accuracy on test data

From predictions in Fig 16, limitations of the model can be observed. There is a tendency for the predictions to accumulate a delta in predicted price over time. Also, the model's ability to predict far into the future is limited. Predictions routines can be modified to predict far into the future, but error rate will have no meaning in those cases.

#### JUSTIFICATION

Overall, the final model aligns with solution expectations and performs significantly better than the benchmark models (study 2, study 3). On testing data, this model has accuracy over 99% across all stocks.

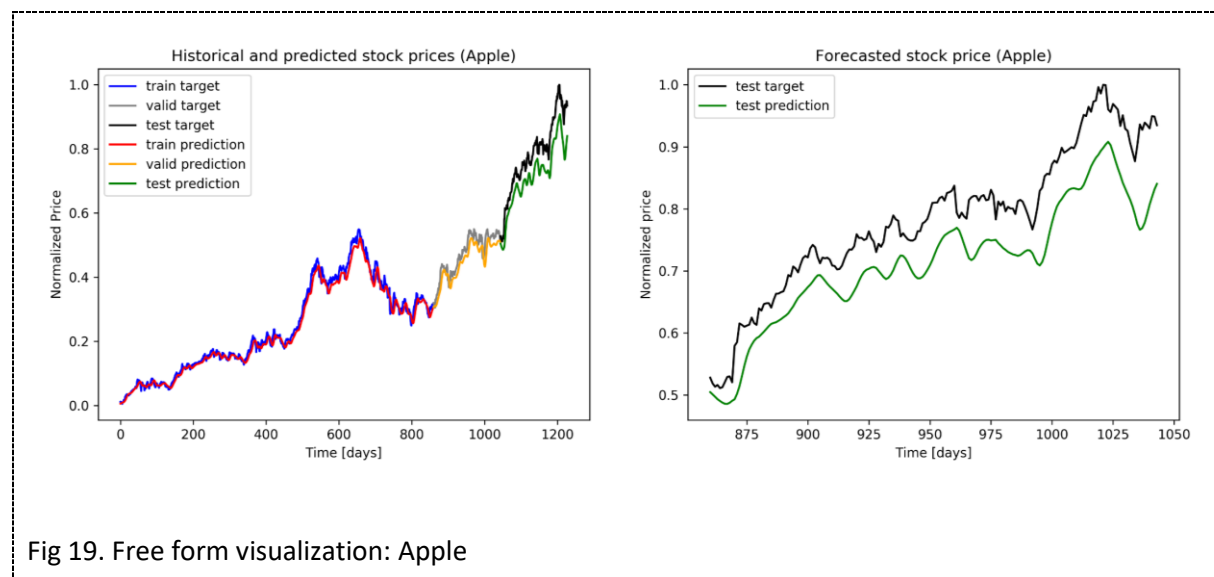
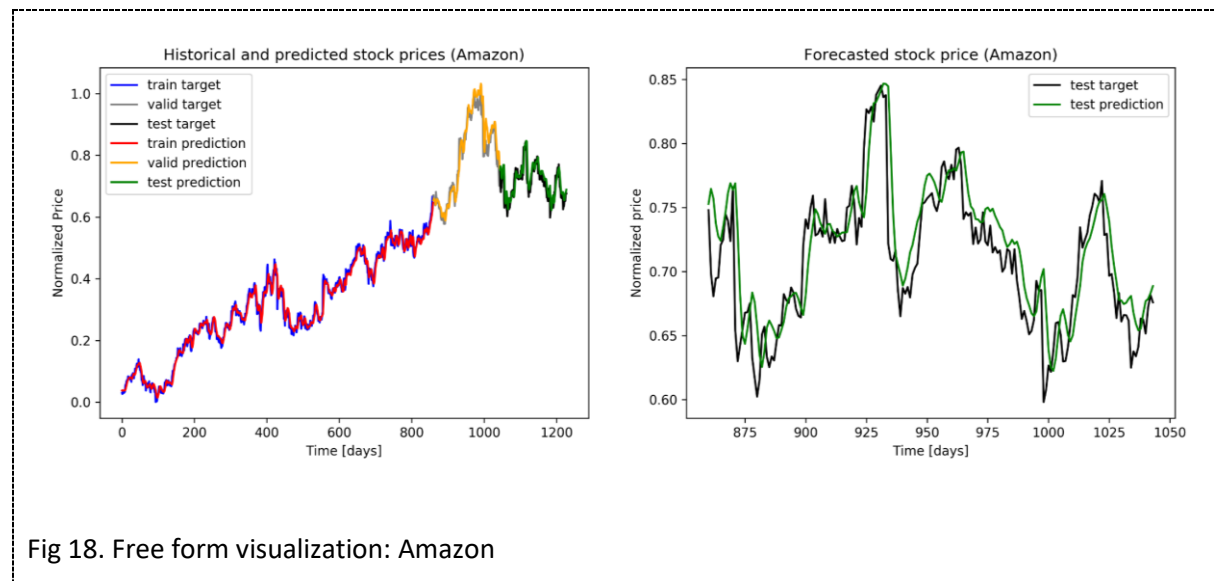
The solution gives a reasonably accurate forecast but needs more fine tuning to improve upon limitations (delta accumulation in predictions over time). A more complete system will require considering factors like transaction cost to qualify for a performant trading tool.

## CONCLUSION

### FREE FORM VISUALIZATION

#### Plotting predictions compared with actual prices

This graph visualises the predictions compared with the actual adjusted close prices. The purpose is to see how predictions vary with actual prices. I picked ~150 data points to visualise because visualising all the points at once does not provide much insight.



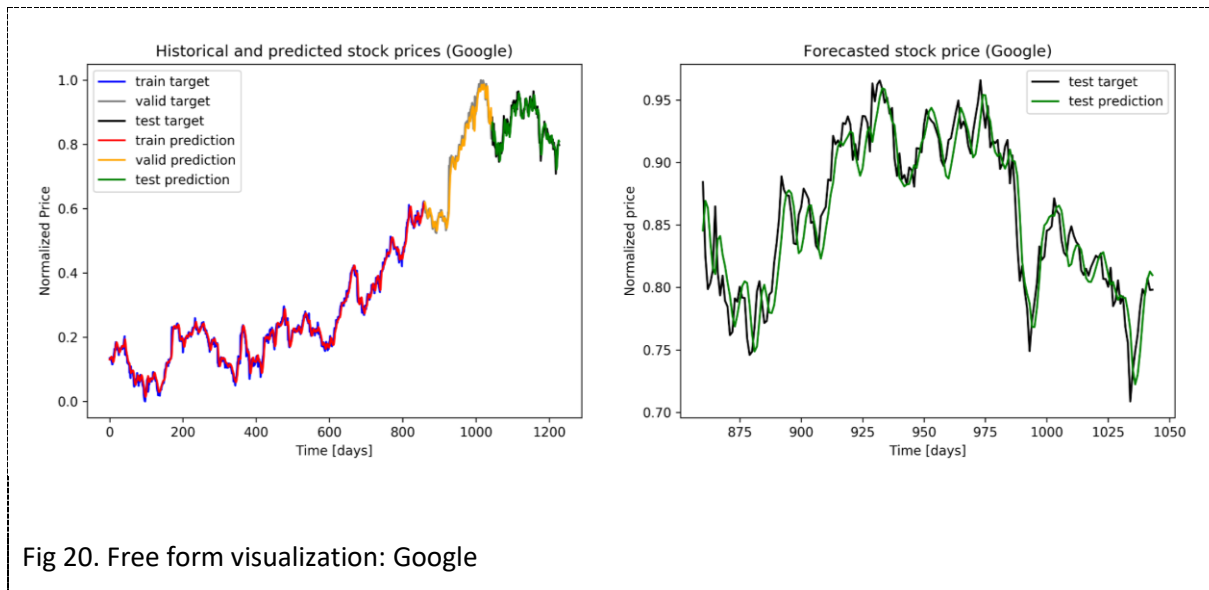


Fig 20. Free form visualization: Google

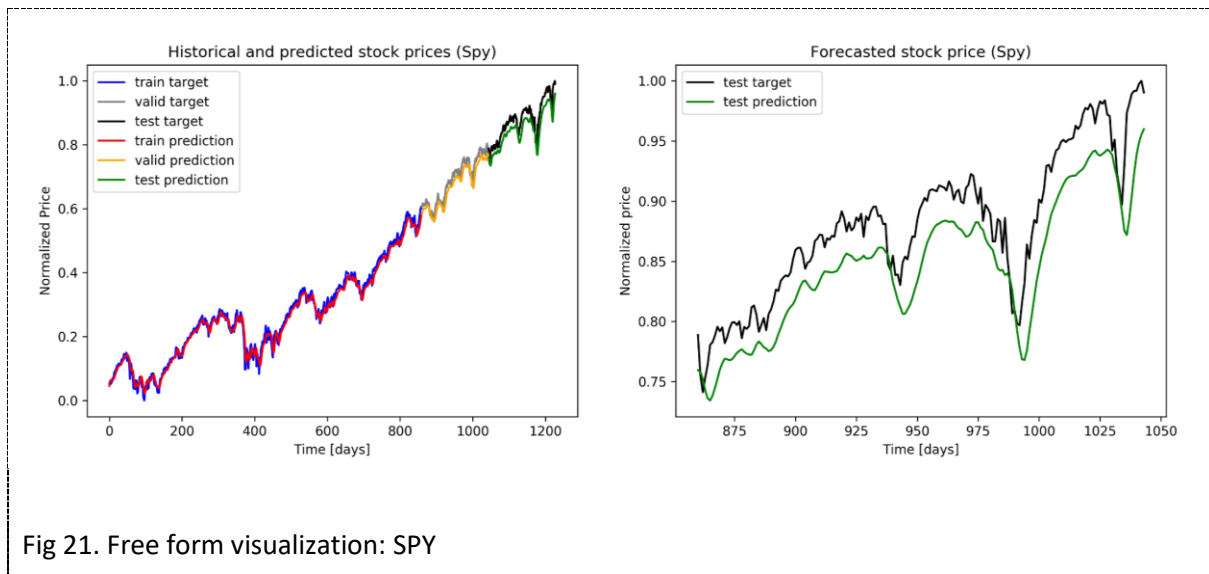


Fig 21. Free form visualization: SPY

## REFLECTION

The overall process I used for developing this project can be summarized in the following key steps

- Access stock data for a date range
  - Download it as a csv from Yahoo! Finance website.
- Pre-process the accumulated data
  - Load the data as panda dataframe using date value as index
  - Normalize the values between 0.0 and 1.0

- Add rolling mean column with a window size of 10
- Create training, testing and validation set
- Create a deep learning model
  - Define the model graph using tensorflow
  - Train the model
- Analyse the model's performance
  - Capture the model's predictions
  - Calculate the error
  - Plot the results

After the initial iterations, I repeated the training with different model architectures and hyperparameters. I found, stacked LSTM performance was overall better for the stocks (based on MSE value). I tested the model on ~150 data points and found few limitations in model (delta accumulation in predictions over time). Also, I realize a more complete system will require considering factors like transaction cost to qualify for a performant trading tool

### *IMPROVEMENTS*

Following improvements can be done to increase the reliability and performance of the model

- Wider selection of features, e.g. Stocks from other stock markets (e.g. BSE), company specific figures such as P/E ratios. This can give a more accurate model.
- Having sentiment for a stock as a feature, based on news events, company's announcements (e.g. dividend) and social media feeds can improve the model performance.
- Obtain and combine data from multiple sources to minimize missing data.
- I would like to create a web app, which can give predictions in real time by crawling stock data.

## REFERENCES

1. [Efficient market hypothesis](#)
2. [Artificial Intelligence: The new frontier for hedge funds](#)
3. [Financial Market Time Series Prediction with Recurrent Neural Networks, Armando Bernal, Sam Fok, Rohit Pidaparthi](#)
4. [Wikipedia article for  \$R^2\$](#)
5. [Wikipedia article for rolling mean](#)
6. [Keras documentation](#)
7. [Time Series Forecasting Based on Augmented Long Short-Term Memory](#)
8.  [\$R^2\$  definition, Boston housing price prediction](#)
9. [LSTM functions](#)
10. [RNNs Machine learning mastery](#)