Stock Market Forecasting using Recurrent Neural Networks in Python

Table of Contents

1. Introduction

The stock market provides a crucial place for corporations to gather equity and facilitate investment within the company. The ability to forecast stock prices is notoriously difficult and can seem unattainable. There are numerous ways of forecasting markets, including time series analysis (e.g. ARIMA). With the recent advent of deep learning, there has been a lot of interest in applying deep learning techniques to predict stock markets. This research project outlines different ways of forecasting the stock market returns using a **Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM)** and **GRU (Gated Recurrent Unit).** In this paper, RNN refers to a vanilla recurrent neural network (i.e. no long-term memory). We will delve into each of these in more detail in the coming chapters.

Using these types of deep learning techniques will enable us to understand the benefits and drawbacks of each in predicting stock markets. We will be using multi-variate data to train the models, providing us insight on how these models learn and predict.

Whilst this research project is predominantly focussed on implementing these models in Python (and thus using readily available libraries), you can also find the mathematics behind neural networks in slides here: https://github.com/k4v1t/Deep_Learning/blob/main/Deep%20Learning.pdf.

It is worth noting that the main purpose of this project is to gain an understanding of how one would go about researching the efficacy of neural networks for stock market prediction. Given the simplicity of the data used in this project, we do not expect to be able to predict stock market returns with great accuracy. It is the process which is more important for this research project.

2. Learning Objectives

- **Understand** the fundamentals of RNN, LSTM and GRU.
- **Acquire and preprocess** historical stock price data.
- **Build** and **evaluate** the neural networks.
- **Forecast** future stock market returns and evaluating the results.
- **Visualising** the outputs effectively.

3. Pre-requisites

- Proficiency in Python programming.
- Familiarity with basic concepts in deep learning – you can find some introductory slides on deep learning using the GitHub link below: https://github.com/k4v1t/Deep_Learning/blob/main/Deep%20Learning.pdf
- Understanding of financial markets and stock price data.
- Libraries: `pandas`, `numpy`, `matplotlib`, `tensorflow`

4. Project Overview

We will:

- **Select** several publicly traded companies.
- **Fetch** historical stock price data.
- **Conduct** data cleaning and preprocessing.
- **Build** and **validate** the neural network.
- **Forecast** future stock returns.
- **Evaluate** the model's performance.
- **Visualise** and **interpret** the results.

5. Module 1: Understanding Recurrent Neural Networks

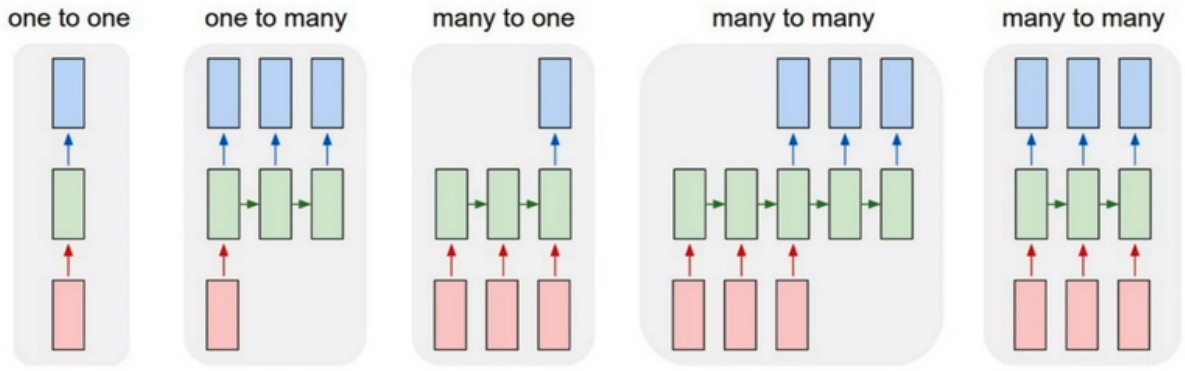
5.1. Vanilla Recurrent Neural Network (RNN)

An artificial neural network has a fixed number of inputs and outputs, which creates a challenge when we our input or output data has a variable size. Another issue with an artificial neural network is that it does not consider the order of the input data. Both are important factors when thinking about translation, autocomplete or time series prediction. An RNN allows us to overcome these drawbacks of an artificial neural network. The overall concept remains like an artificial neural network: using weights, biases and non-linear activation functions for training and prediction.

There are four main types of RNNs:

(1) One-to-one: One input to one output
(2) One-to-many: One input to many outputs, e.g. baby name generator which takes in the biological sex as an input
(3) Many-to-one: Many inputs to one output, e.g. converting a text review to a star rating
(4) Many-to-many: Many inputs to many outputs, and there are two main sub-types:
    a. Same input and output length, e.g. entity recognition
    b. Different input and output lengths, e.g. translation

The exhibit below provides an illustration of how each of these types of RNN work in practice.



The red boxes here are the inputs, with the blue boxes being the outputs and the green boxes the actual RNN operation. Each of these green boxes will contain one or many RNN cells. Each cell takes in some input data, applies the weights and biases, followed by a non-linear activation function to calculate a "state", and passes it to the next cell and/or output (depending on the type of RNN). This process is known is **forward propagation**, and this is used to predict the output of an RNN. The mathematical details of this forward propagation can be found in the PDF mentioned previously.

A fair question to ask next would be "how does the RNN know the nuances of a set of input data to predict an output?". This is where **backward propagation** (or **backpropagation**) comes in, and it is how we "train" an RNN to "learn" the nuances of out input data. In essence, backpropagation is the process via which we calibrate the weights and biases with respect to the use case. Again, the mathematical details of backpropagation can be found in the PDF previously mentioned. For the purposes of this research project, we will be using readily available `tensorflow` libraries to train our RNNs.

One major drawback of an RNN is the concept of a **vanishing gradient**. Without going into the mathematical details, it essentially refers to the problem of an RNN "forgetting" the input information from a long time ago. To overcome this drawback of an RNN, we can use Long Short Term Memory (LSTM), which allows the neural network to retain relevant information from the past.

5.2. Long Short Term Memory (LSTM)

An LSTM provides a way to get around the vanishing gradient problem in an RNN. Whilst an RNN only passes one state to the next cell, an LSTM passes two states to each cell. These are the hidden state (the short term memory – same as the RNN) and the cell state (the long term memory). These states are calculated using several "gates", which determine what information gets retained and passes on to the next cell. Let us define these gates below:

- **Forget Gate**: This gate determines whether we need to discard any information.
- **Input Gate**: This gate tells the LSTM there is new information.
- **Output Gate**: This gate dictates how much to reveal to the next cell.

We can then use these gates in a way that the LSTM can understand long term dependencies whilst not being overloaded with data from all previous cells.

These new gates provide us additional parameters to calibrate. Whilst these additional parameters allow the model to gain more understanding of our data, it makes the training process slower and computationally more resource intensive. If there is a need for a faster and more computationally efficient model, we can use a Gated Recurrent Unit (GRU) instead of LSTM.

5.3. Gated Recurrent Unit (GRU)

A GRU can be thought of as a simplified version of LSTM which provides a good balance between accuracy and computational efficiency. For larger datasets and higher accuracy, LSTM provide a better result. In contrast, smaller datasets and limited

computational resources lend themselves to a GRU. Whilst LSTM has three gates, a GRU only has two:

- **Update Gate**: This gate determines how much of the past needs to be retained.
- **Reset Gate**: This gate decides how much of the past is required to compute the present.

Now that we have a high-level understanding of these 3 deep learning models, we can move on to data gathering and processing.

6. Module 2: Data Acquisition

The stock market data used for this research project can be found on the Kaggle link below:
https://www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs/data

We can find the daily open, high, low, close prices and volume traded for all US-based stocks and ETFs traded on the NYSE, NASDAQ and NYSE MKT. Whilst the data was last updated in 2017, it still provides enough information for this research project. For this project, we will use AAPL (Apple) and HPQ (Hewlett-Packard) stock data.

7. Module 3: Preprocessing and Feature Generation

For each stock, we need to convert the price data into log-returns which will normalise the data, lending itself to be used more easily in the neural networks. When it comes to volume, depending on the goal of the exercise, it can be normalised in several ways:

(1) Z-Score: This is ideal for general modelling purposes as it provides stationarity and allows us to keep relative scale across different stocks. However, it does assume that the variance of the volume remains stable, which might not hold in very volatile markets.
(2) Log or Log-Difference: This measure can be useful if volume spans multiple orders of magnitude. It allows us to handle skewed distributions and captures relative changes. However, it still exhibits non-stationarity which is not ideal for training purposes.
(3) % Change: Whilst this removes some of the non-stationarity in the data, it can still be quite noisy during times of low-volume trading.
(4) Rolling Normalisation: We can use a rolling mean and variance to z-score the volume over time. This allows the model to adapt to changes in regimes.

Given we will be using a rolling exponentially weighted moving average (EWMA) to create features of these variables, a simple z-score will likely be fine for our purposes.

As mentioned, it will be the EWMA versions of these metrics which will be provided as the independent variables for the training of the neural networks. The EWMAs will be calculated for half-lives of 1D, 5D, 10D and 20D. As such we will have 20 independent variables (4 EWMA versions of open, high, low, close and volume each).

Once these 20 independent variables have been estimated, the next step is to create a training, validation and test set for the neural networks. We will use the neural networks

to predict the stock return over 3 time horizons: 1 day, 5 days and 10 days. For predicting the 1-day return, we will use the independent variables from the previous 5 days. For predicting the 5-day return, we will use the independent variables from the previous 10 days. Finally, for predicting the 10-day return, we will use the independent variables from the previous 20 days. So, we will have 3 sets of training/validation/test datasets, one each of the different time horizons.

For the target variable, we will use the log-return of close prices for each stock. When predicting over multiple days, we will use the return over that time horizon. For example, prediction of a 5-day return means the return over the 5 days rather than 5 returns over each of the next 5 days. For training purposes, this target variable will also be normalised using `StandardScaler`.

8. Module 4: Return Prediction: Architecture

Before we move to forecasting returns for the two stocks over different horizons, it is important to highlight the architecture we will use for the different neural networks along with the hyperparameters for these neural networks. When predicting using the neural networks, we will use the Mean Absolute Error (MAE) to train and test the efficacy of the prediction. The overall dataset will also be split into train, validation and test to avoid common pitfalls like overfitting and biased evaluation.

8.1. Input Layer

The input layer for all 3 models will consist of 128 units of RNN/LSTM/GRU. To avoid overfitting, especially due to noisy market data, this layer also introduces a dropout and recurrent dropout. These will prevent any over-reliance on specific features by randomly using zero inputs. The value of dropout and recurrent dropout used is 0.2.

8.2. Dense Layers

The architecture consists of 4 dense layers, which compressing progressively as we go from input to output (64 to 32 to 16 to 8). These 4 layers allow more capacity to model complex non-linear functions. The progressive compression enhances the networks' ability to learn the hierarchies of features.

However, more layers do increase the risk of overfitting. To mitigate this risk, we will use an L1 regularisation in each of the dense layers with a lambda of 0.001. This encourages sparse weights and can be treated as a feature selection approach. Given the high dimensionality of our feature space, this can be helpful in reducing overfitting. The use of L1 over L2 regularisation is beneficial given the need to ignore unimportant features rather than smoothly regularising all weights.

8.3. Output Layer

Finally, the neural network provides the output which is just one number. If we are predicting returns, it'll simply be the predicted return. However, if we wish to predict a probability then we will need to use a sigmoid function at the output layer.

9. Module 5: Return Prediction: Results

Let us begin by attempting to forecast the stock returns over 3 different horizons of the two companies using the 3 models. This module will go through this for the two stocks, 3 time horizons and the three models specified previously. For the user's benefit, the notebooks used have the same name as the sub-sections below. The results will be summarised in the final sub-section of this module.

9.1. Return_HPQ_5D_1D

Stock: HPQ

Window Size: 5 Days

Prediction Horizon: 1 Day

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

HPQ_5D_1D	Baseline	RNN	LSTM	GRU
Train	1.6618	1.6622	1.6624	1.6624
Validation	1.5985	1.5985	1.5985	1.5985
Test	1.2548	1.2545	1.2544	1.2544

9.2. Return_HPQ_10D_5D

Stock: HPQ

Window Size: 10 Days

Prediction Horizon: 5 Days

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

HPQ_10D_5D	Baseline	RNN	LSTM	GRU
Train	3.7742	3.7734	3.7734	3.7734
Validation	3.6844	3.6828	3.6828	3.6828
Test	3.0761	3.0599	3.0584	3.0592

9.3. Return_HPQ_20D_10D

Stock: HPQ

Window Size: 20 Days

Prediction Horizon: 10 Days

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

HPQ_20D_10D	Baseline	RNN	LSTM	GRU
Train	5.2203	5.2153	5.2144	5.2145
Validation	5.0992	5.0964	5.0987	5.0979
Test	4.5862	4.5562	4.5428	4.5469

9.4. Return_AAPL_5D_1D

Stock: AAPL

Window Size: 5 Days

Prediction Horizon: 1 Day

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

AAPL_5D_1D	Baseline	RNN	LSTM	GRU
Train	2.1470	2.1474	2.1479	2.1473
Validation	1.2400	1.2397	1.2396	1.2397
Test	1.0216	1.0207	1.0198	1.0208

9.5. Return_AAPL_10D_5D

Stock: AAPL

Window Size: 10 Days

Prediction Horizon: 5 Days

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

AAPL_10D_5D	Baseline	RNN	LSTM	GRU
Train	4.9204	4.9033	4.9030	4.9030
Validation	3.1108	3.0888	3.0888	3.0888
Test	2.4459	2.4049	2.4036	2.4036

9.6. Return_AAPL_20D_10D

Stock: AAPL

Window Size: 20 Days

Prediction Horizon: 10 Days

Using the feature space generated for these parameters, we get the following output for the MAE per neural network, along with the baseline MAE.

AAPL_20D_10D	Baseline	RNN	LSTM	GRU
Train	7.1340	7.0900	7.0900	7.0900
Validation	4.5273	4.4653	4.4653	4.4653
Test	3.5848	3.4801	3.4810	3.4809

9.7. Return Prediction Summary

The results from the two stocks across different time horizons and models are summarised below:

- *Across the 1-day prediction horizon, none of the models fare any better than the Baseline for both stocks*. This makes sense as daily returns are extremely noisy can be driven by:
    - Random market microstructure noise.
    - Headlines, sentiments and tweets.
    - Liquidity effect or unwinding positions.

  Additionally, when optimising a loss function such as MAE on 1-day returns, the targets can be close to zero and the *model learns that predicting zero is a safe bet*. This means it performs the same as the Baseline MAE, as we can see in the results above.

- When we review the models across 5- and 10-day horizons, we see an improvement in the MAE compared to the Baseline. This improvement is seen across train, validation and test datasets. This means we are not overfitting to the training data and the model is able to improve on the previously "unseen" test data. The reason for this improvement across these longer time horizons can be explained as follows:
    - Longer-term horizons tend to reflect more meaningful trends such as momentum, mean reversion and fundamentals – the model has a better chance of picking up an actual signal.
    - Over longer periods, noise cancels out, and the cumulative return starts to reflect real drivers of price movement.
    - Over 5-10 days, EWM trends, moving averages and price-volume relationships start to matter, giving the features more predictive power.

Whilst some of the models work better than the Baseline across longer time horizons, none of them offer a significant improvement. This is due to the simplicity of the data used in our feature space. Even though we have converted the original data into EWM versions to incorporate short- and long-term behaviour, we are still only using the stock's open, high, low, close and volume data. A stock's return depends on a myriad of other things, including fundamentals, news, market/beta returns, etc. As mentioned previously, the focus of this research project is on process rather than prediction power.

10. Module 6: Probability Prediction: Architecture

Another approach to forecasting returns is to predict the probability of a positive return over the 1-, 5- or 10-day prediction horizon. This will turn the problem from a regression to a binary classification, something which is often more learnable for neural networks. It can also be used as a trading signal more easily as probabilities are more actionable.

However, before we go into the results of this exercise, we need to address the change in architecture required given we're now trying to solve a different problem. These changes are summarised below:

- Given the binary classification, there is less complexity required from the model. This is because the model only needs to learn a decision boundary between two groups. As such, we can collapse the network down in length and depth to avoid overfitting to noise. *The architecture for this exercise will have one input layer with 16 units, one dense layer with 8 units and one output layer with 1 unit using a sigmoid activation function*.
- Whilst regularisation still works for binary classification, *we will remove any regularisation* given the architecture of our neural networks to avoid the model just shrinking too many weights.
- In return prediction the *optimiser* used was Adam. Whilst this worked well enough in the previous use case, using *RMSprop* yielded better results for the binary classification, and we will use that.
- We will also now have to consider class weights when training the neural network. If one class dominates (e.g. 70% of days have a positive return), the model may realise that it can "cheat" by simply predicting the majority class. To avoid this, we can also provide it "class weights", which tells the optimiser that a mistake in a minority class if more costly.
- The model outputs probabilities between 0 and 1, but we need to decide what counts as positive. This is called the "threshold" and by default this is 0.5. However, depending on how conservative the model is, we might need to change this threshold. In this case, we will tune the threshold which trades off between precision and recall (these terms are explained in detail below).

Finally, when reviewing the efficacy of the model, we need to output metrics which are explained below.

Metric	***Meaning***	***Interpretation***
Accuracy	Overall correct predictions / total samples	Can be misleading with imbalance
Precision	Of all predicted positives, how many were correct?	High precision = few false positives
Recall	Of all actual positives, how many did we correctly predict?	High recall = few false negatives
F1-score	Harmonic mean of precision and recall	Best when you care about balance
AUC	How well can the model rank positives above negatives (threshold-independent)	Best for imbalanced datasets

11. Module 7: Probability Prediction: Results

Let us begin by attempting to forecast the stock returns over 3 different horizons of the two companies using the 3 models. This module will go through this for the two stocks, 3 time horizons and the three models specified previously. For the user's benefit, the notebooks used have the same name as the sub-sections below. The results will be summarised in the final sub-section of this module using the model prediction for the test data.

11.1. Prob_HPQ_5D_1D

Stock: HPQ

Window Size: 5 Days

Prediction Horizon: 1 Day

Model	**Accuracy**	**Precision (0)**	**Recall (0)**	**F1 (0)**	**Precision (1)**	**Recall (1)**	**F1 (1)**	**AUC**
RNN	0.5104	0.4878	0.6300	0.5499	0.5460	0.4022	0.4632	0.5161
LSTM	0.4963	0.4837	0.9040	0.6302	0.5956	0.1278	0.2104	0.5159
GRU	0.5311	0.5340	0.0960	0.1627	0.5308	0.9243	0.6743	0.5101

11.2. Prob_HPQ_10D_5D

Stock: HPQ

Window Size: 10 Days

Prediction Horizon: 5 Days

Model	**Accuracy**	**Precision (0)**	**Recall (0)**	**F1 (0)**	**Precision (1)**	**Recall (1)**	**F1 (1)**	**AUC**
RNN	0.4801	0.4393	0.7280	0.5479	0.5836	0.2909	0.3883	0.5095
LSTM	0.4735	0.4486	0.9444	0.6083	0.7290	0.1140	0.1972	0.5292
GRU	0.4660	0.4434	0.9157	0.5975	0.6562	0.1228	0.2069	0.5193

11.3. Prob_HPQ_20D_10D

Stock: HPQ

Window Size: 20 Days

Prediction Horizon: 10 Days

Model	**Accuracy**	**Precision (0)**	**Recall (0)**	**F1 (0)**	**Precision (1)**	**Recall (1)**	**F1 (1)**	**AUC**
RNN	0.4332	0.4303	0.9903	0.5999	0.6667	0.0145	0.0284	0.5024
LSTM	0.4581	0.4413	0.9884	0.6101	0.8723	0.0596	0.1116	0.5240
GRU	0.4631	0.4404	0.9284	0.5974	0.6783	0.1134	0.1943	0.5209

11.4. Prob_AAPL_5D_1D

Stock: AAPL

Window Size: 5 Days

Prediction Horizon: 1 Day

Model	Accuracy	Precision (0)	Recall (0)	F1 (0)	Precision (1)	Recall (1)	F1 (1)	AUC
RNN	0.4976	0.4822	0.8207	0.6075	0.5617	0.2068	0.3023	0.5130
LSTM	0.5024	0.4859	0.8687	0.6232	0.5938	0.1727	0.2676	0.5207
GRU	0.5156	0.4935	0.8611	0.6274	0.6207	0.2045	0.3077	0.5328

11.5. Prob_AAPL_10D_5D

Stock: AAPL

Window Size: 10 Days

Prediction Horizon: 5 Days

Model	Accuracy	Precision (0)	Recall (0)	F1 (0)	Precision (1)	Recall (1)	F1 (1)	AUC
RNN	0.4647	0.4405	0.9328	0.5984	0.6962	0.1151	0.1975	0.5239
LSTM	0.5138	0.4580	0.7479	0.5681	0.6429	0.3389	0.4438	0.5434
GRU	0.4359	0.4300	0.9804	0.5978	0.6667	0.0293	0.0561	0.5048

11.6. Prob_AAPL_20D_10D

Stock: AAPL

Window Size: 20 Days

Prediction Horizon: 10 Days

Model	Accuracy	Precision (0)	Recall (0)	F1 (0)	Precision (1)	Recall (1)	F1 (1)	AUC
RNN	0.5635	0.4690	0.5828	0.5198	0.6594	0.5504	0.6000	0.5666
LSTM	0.6019	0.8000	0.0237	0.0460	0.5995	0.9960	0.7485	0.5098
GRU	0.5743	0.4830	0.7130	0.5759	0.7104	0.4798	0.5728	0.5964

11.7. Probability Prediction Summary

The results from the two stocks across different time horizons and models are summarised below:

- Longer prediction horizon consistently improves positive return recall and F1. We can see that the 1-day prediction has low positive return recall for both stocks. However, when looking at a 10-day horizon, the models perform much

better, with the LSTM one providing a 0.75 F1 for AAPL. Like the return prediction, longer horizons mean less noise and more signal strength. We also see this result when looking at the AUC (going from 0.51 to 0.60)

- We see that GRU is the most consistently balanced model, providing the best F1 and trade-off between positive and non-positive returns. This is because it tends to avoid extreme bias toward a single class that RNN and LSTM typically suffer from.
- If we are interested in detecting positive momentum, i.e. maximising the prediction of positive returns, LSTM does the best job especially over longer-term horizons. However, this often comes with a cost of worse performance in classifying a non-positive return.
- Like the return prediction, AAPL seems to lend itself to better prediction compared to HPQ. This could be due to a lower trend consistency or noisier structure of HPQ.
- Over short-term horizons, an inflated accuracy can mask the weak positive or non-positive performance or extreme class recall imbalance.