Convolutional Neural Network towards image classification of the CIFAR-10 dataset

Julien Guinot
The University of Adelaide
julien.guinot@student.adelaide.edu.au

Abstract

Stock price prediction is of high interest to daytraders and stockbrokers. predicting such a complex ecosystem has long been a studied problem, and the advent of machine learning has made leaps and bounds in such predictions.

Recurrent Neural networks in particular (RNNs) have proved to be proficient in predicting the stock market through various architectures. This paper presents an exploratory analysis of various recurrent Neural Network architectures with the goal to predict google stock prices based on historical time series data.

The basics of Recurrent Neural Networks as well as more modern GRU and LSTM are presented, and three models are trained with various hyperparameter configurations. A final, proficient model is exhibited, presenting less that 1% mean absolute error on the test set derived from the proposed dataset. Other models are evaluated and proven to be less proficient.

1. Introduction

Predicting stock prices is a dream for any stockbroker. An accurate prediction of the evolution of stock prices and the volume of exchange over any given period provides an opportunity to make a lot of many for anyone with the right algorithm. Sequential machine learning models such as recurrent neural networks (RNN) have been used repeatedly to attempt tackling this problem [1], [2], [3], among their other uses. With a varying degree of success which has been growing over the course of recent years, market prediction algorithms have led to the widespread adoption of trading bots for day traders to use as part of their income source for example.

This paper presents an approach to stock value prediction using recurrent neural networks, including the exploration of various avilable well-known models : Gated Recurrent Unit (GRU), Long short term Memory (LSTM) and vanilla RNNs. Through the analysis of the provided data and selection of a model, we train a model of satisfactory performance for stock price prediction, with the objective of the given algorithm presenting acceptable usability for any one wishing to use it.

Firstly, this paper will provide an overview of the dataset at hand for the task of stock price prediction. In a second section, the mathematical principles behind recurrent neural networks, as well as better performing variants (LSTM, GRU, bidirectional, stacked) will be presented. After analysis of the available models and the task at hand, an appropriate RNN architecture will be selected and trained on the provided data for multiple RNN variants. After converging on appropriate hyperparameters, final results will be presented, analyzed, and will serve towards selecting the best possible model for stock prediction on the provided data.

2. Overview of provided data

The objective of the project is to build an algorithm to predict stock prices for a given stock, from that given stock's historical data It is a common regression task within machine learning. As such, the data used must be in the form of a time series containing features which will be fed to the algorithm to predict the future price.

The data used for stock prediction is Historic GOOGL stock data from 2012/01/03 to 2016/12/30. The dataset, which can be found at `https://www.kaggle.com/rahulsah06/gooogle-stock-price`, contains daily values for The following features:

- **Open price:** This is the price at which each stock unit is listed at the opening of the stock market on any given day *(9.30 Eastern Time for the New York Stock Exchange (NYSE)*
- **Close price:** This is the price of each stock unit at closing time of the NYSE
- **High Price and Low Price:** The highest price attained by the stock on the given day and the lowest price attained by the stock on the given day

- **Volume:** The amount of stock units traded (bought and sold) on the given day

Which are split into a 1258 day-long training set and a 20 day-long validation set. The time series of these values are shown below (Fig. 1). On the left are shown stock price values over the timespan of the dataset, and on the right is shown the Volume feature over time.

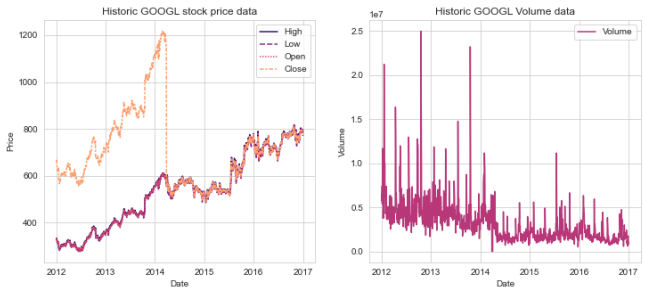

Figure 1: Dataset feature evolution over time

Some things are to be immediately noticed:

- There is a sudden drop in Close price surrounding march 27,2014. Though this might seem surprising, this is due to google performing a 2:1 split of their stocks on this given date, explaining the halving of closing price (`https://www.stocksplithistory.com/alphabet/`). However, this means that Close price might be a misleading feature for our algorithm if we want it to learn somewhat long sequences.

- Volume data and price data present drastically different ranges, with Volume data attaining 10^7, and prince data staying within the thousands range.

To address these discrepancies within the data which might lead our model to take much longer to converge, we apply two simple preprocessing steps:

- Drop the close price column from our training data

- **Apply min-max scaling to the training and test data so that all values are contained within the 0-1 range** (Common preprocessing technique for stock value prediction)

With this basic preprocessing applied, the data is perfectly suited to the task at hand. Possible additional feature generational will be addressed in the bonus section.

3. Recurrent Neural Networks

Recurrent Neural Networks (RNN) are Networks designed for learning sequential data. This data can take many shapes (sequence of frames from a video, sequence of words from a language, sequence of values from a time series). Their uses are plentiful, as they can be applied to all sorts of Natural Language Processing tasks (translation, sentiment analysis, image captioning, speech recognition), in time series prediction, in machine learning applied to musical sequences.

RNNs were first described in [4], and have since grown to include more complex architectures such as LSTM and GRUs, bidirectional and stacked networks, and memory and attention mechanisms. This section showcases the principles behind the basic building blocks of these architectures.

3.1. RNN principles

RNNs present a similar structure to neural networks in that they are comprised of neurons which compute the output for each step of the given sequence. The general structure of a RNN is shown below (Fig. 2):

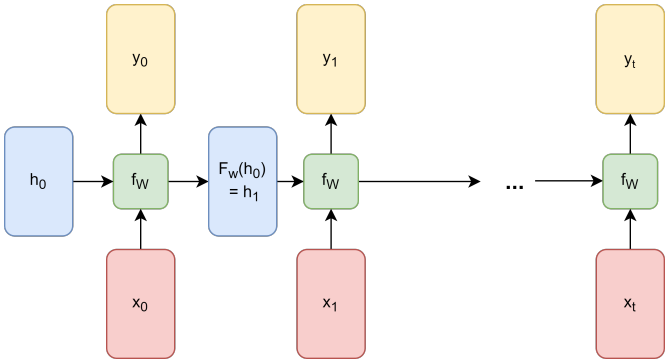

Figure 2: General structure of a RNN

In the above figure, the hidden state h_t represents a state computed by the network given past historical data. x_t is the input vector at iteration t of the sequence (for instance, the value of a time series). y_t is the output we aim to have the network be able to predict. The parameters the network learns are contained within the function f_W.

Indeed, at each step, the function applied to the previous historical data is the same, and it is the parameters of this function that the network learns. More specifically, if we notate W_xh the weights applied to x_t to obtain h_{t+1}, W_hh the weights applied to h_t, and W_{hy} the weights applied to h_t to obtain y_t, the model will learn the parameters W_hh, W_hy and W_hx: The forward propagation through the network can be calculated recursively thanks to the following formula for a vanilla RNN:

$$h_t = \tanh W_{hh} h_{t-1} + W_{x_h} x_t \tag{1}$$

$$y_t = W_{yh} h_t \tag{2}$$

And backpropagation is conducted by computing loss between y_t and the ground truth at each step. Thus, learning is achieved.

There are three types of tasks for recurrent neural networks:

- **Many to one : prediction of a single value from a sequence** (Time series prediction, sentiment prediction, video classification)
- **Many to many : predicting future values from series** (Translation, time series sequence prediction)
- **One to many: single input vector or array into a sequence of outputs** (Text generation from topic, Image captioning).

The task depends on the project, and the appropriate task for our project will be discussed while considering which architecture to build in section **5.2**. The architecture necessary for each class is shown below (Fig. 3).

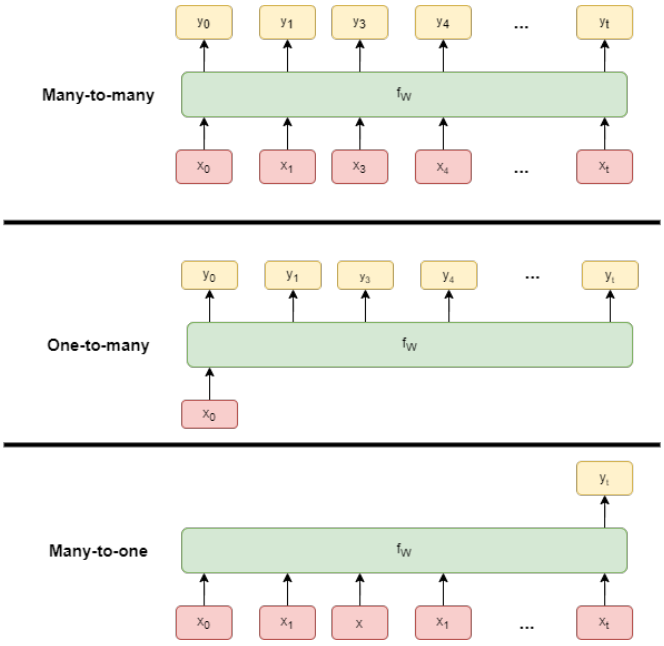

Figure 3: Possible tasks for RNN models

3.2. Improving upon vanilla RNN : LSTM and GRU

Though RNN can prove useful for learning sequential models, they present a few problems. Backpropagating through time for long sequences can induce vanishing and exploding gradients, as many factors of $W_x h$ are taken into account.

Furthermore, Vanilla RNNs also present the problem of forgetting meaningful patterns over very long sequences. More modern RNNs, such as LSTM and GRU, introduce gating blocks for meaningful features in order to bypass neurons much like ResNet does for CNNs. This allows bypassing vanishing gradients, and thus forgetting important features. This section focuses on these more modern RNNs and their principles. Attention mechanisms and stacked RNNs will also be explored.

3.2.1 Gated Recurrent Unit RNN : GRU

Gated Recurrent Unit RNNs, first introduced in [5], introduce the notion of gating to recurrent neural network, which helps in deciding whether or not part of or the entirety of the previous state and the input are used.

The GRU is comprised of an update gate and a reset gate which are represented in the following schematic (Fig. 4):

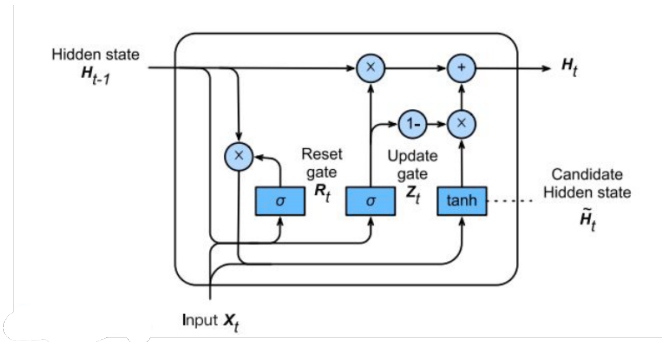

Figure 4: Structure of a GRU cell

And the above cell update scheme can be summarized by the following set of equations:

$$\begin{aligned} r_t &= \sigma(W_{xr} x_t + W_{hr} h_{t-1} + b_r) \\ z_t &= \sigma(W_{xz} x_t + W_{hz} h_{t-1} + b_z) \\ \bar{h}_t &= \tanh W_{xh} x_t + W_{hh}(r_t * h_{t-1}) + b_h \\ h_t &= z_t * h_{t-1} + (1 - z_t) * \bar{h}_t \end{aligned} \tag{3}$$

Essentially, GRU introduce two new mechanisms:

- **Update Gate**: the update gate Z controls how much the past state and input matter at this iteration. If this is close to zero, only current information is stored in the state
- **Reset Gate**: The reset gate R controls which data the model forgets, allowing it to forget irrelevant information

The main strength of GRUs compared to LSTMs are the lack of parameters to train, making for rapidly-trained models and the possibility to train much deeper models.

3.2.2 Long Short Term Memory RNN : LSTM

Long short Term Memory RNNs were introduced in [6] as a solution to vanishing and exploding gradients while training vanilla RNNs on very long sequences. Due to backrpopagating through time with many W factors to multiply by, long sequences are often very hard to train RNN on due to the weights vanishing or exploding during training. It introduces three cells which contribute to the classic RNN cell:

- **Forget cell**: Shrinks input values to zero, helps to forget irrelevant data to the learning process
- **Input gate**: Decides to drop or keep the input data for the current state
- **Output gate**: Decides whether or not the output hidden state is used for the output generated by the LSTM

The contribution of these cells and gates is shown in the following figure (Fig. 5):

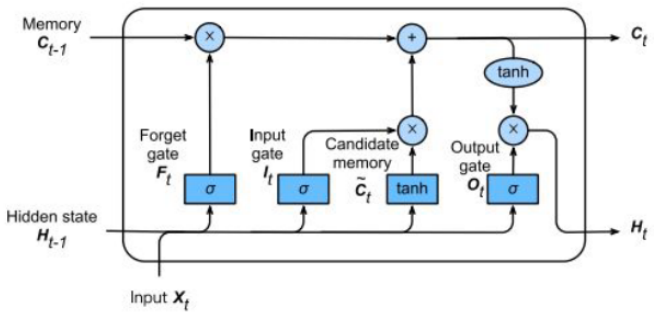

Figure 5: Structure of a LSTM cell

And the state of the output memory C_t and hidden state H_t are given by the following set of equations which summarize the previous schematic:

$$\begin{aligned} I_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ F_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ O_t &= \sigma(X_tW_{xo} + h_{t-1}W_{ho} + b_o) \\ \bar{C}_t &= \tanh X_tW_{xc} + h_{t-1}W_{hc} + b_c \\ C_t &= F_t * C_{t-1} + I_t\bar{C}_t \\ h_t &= O_t * \tanh C_t \end{aligned} \tag{4}$$

Compared to RNNs and GRUs, the LSTM structure is much more complex and thus has more parameters to train. Thus, it should be better at learning subtle features in long term sequences additionally to being able to alleviate the vanishing and exploding gradients problem. However, having more parameters to train, it is more computationally expensive to set up a deep LSTM network than a deep GRU network

3.2.3 Stacked, bidirectional RNNs

Other variants of RNNs exist, such as stacked and bidirectional LSTMs, stacked LSTMs increasing the depth of the network, and bidirectional LSTMS also taking into account the future values of the sequence. THis section focuses on briefly presenting them.

Stacked RNN

Stacked RNNs are the alternative for RNNs to deep neural networks. By passing the hidden state values from one RNN to another stacked on top of it, the model learns equivalent of feature maps from one time sequence to another. This apparatus is shown in figure 6 below.

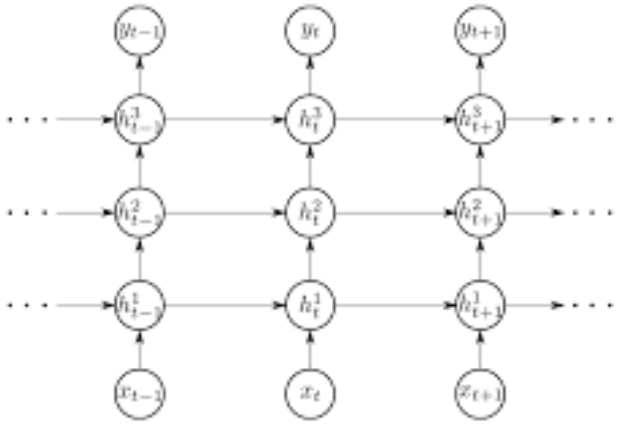

Figure 6: General structure of a stacked RNN

The main advantage of stacking RNNs being that it increases model complexity, and thus allows for learning more complex patterns. However, it also increases computation time greatly.

Bidirectionnal RNNs

Bidirectionnal RNNs were implemented to better model sequences where context is important (and known). For instance, language translation tasks need the whole sentence to translate correctly, video frame interpolation likewise.

A bidirectionnal LSTM takes into account both past and future inputs for training. This can be seen in the following figure of the general structure of a bidirectional RNN (Fig. 7)

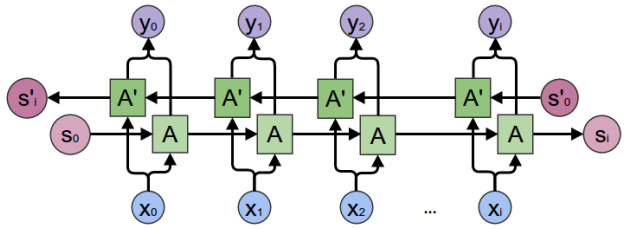

Figure 7: Bidirectional RNN structure

3.2.4 Memory and attention

4. Code

All the code for the visialuzations shown in the following sections as well as the source code used to generate RNN models and train them can be found at the following github repository:

`https://github.com/Pliploop/Stock_Prediction_RNN`

5. Training and performance analysis

This section will focus on training our own RNN to predict the stock market by using the dataset described in section 2. The considerations of the task at hand and their influence on our choices for data generation will be discussed in a first subsection, as well as our considerations for which architectures to train.

Concerning traning, we will be discussing performance metrics for regression tasks, constructing a baseline training curve from our base task consideration. Finally, we will hone in on a set of hyperparameters presenting satisfactory performance and comparing all our trained models.

A last section will have as focus metric comparison of all of the previous models, as well as some failed training attempts with mediocre hyperparameters.

5.1. Task

Firstly, to correctly identify the training data needed for the task at hand as well as the appropriate architecture for our model, we need to correctly define exactly the task at hand.

Our goal for this project is usability by a trader who wishes to predict the stock market to **make money**. So, our goal is going to be predicting the price of the GOOGL stock market from a sequence of length M (M=30 corresponding to a training history of a month), for only one day, and for only the Open price (which is when a trader will be selling)

So, our baseline task is going to be to predict the next **one day Open price** from a training history of **30 days**. This categorizes our task as a one-to-many regression task.

5.2. Data splitting and overlap generation

The original dataset presents a training-test split of 1258/20. So, the data for 20 days is contained within the test set and the data for the remaining 1258 days is contained within the training set.

To fall back into common and tried-and-true splitting methods, this represents a **98.5%-1.5%** training-test split, which we do not consider to be appropriate compared to the usual **80-10-10** train-test-validation split, for instance.

Not only is there no validation data contained in the out-of-the-box dataset, but the test set is much too small, and does not allow for 30-day history prediction. So, we decide to re-split the data into train, test and validation sets for all features **High,Low,Open,Volume**. The train-test-validation split is shown below in Fig. 8

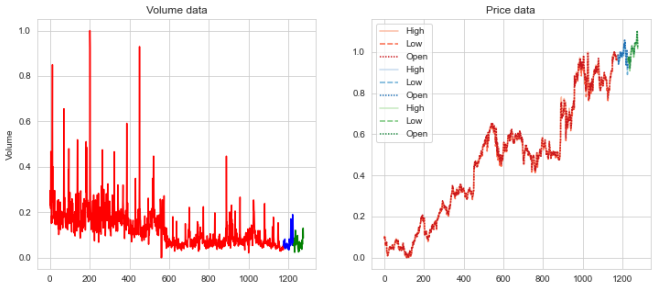

Figure 8: Train-test validation split from the original dataset

This represents a **1178-50-50** train-test-validation split, or a **92%-4%-4%** split. Though the test and validation proportions remain small compared to the usual split, it is generally recommended to use more training data for training recurrent neural networks, as these are generally prone to underfitting for long sequences.

Sliding window dataset generation

Furthermore, instead of generating one 30-day sequence of training history every 30 days, implementing a sliding window to determine the training set for each predictable day provides much more training data. So, we decide to implement a **30-day sliding window** over the entirety of the training set, providing us with 1148 training samples instead of 39.

5.3. Architecture Overview

Now that a appropriate dataset has been generated for training our model, we must decide the architecture of the trained model. All previously presented models will be trained with the exception of bidirectional RNNs:

- **vanilla RNN**
- **LSTM**
- **GRU**
- **Stacked RNNs**

We choose not to implement bidirectional RNNs because they are irrelevant in a real use case of the task at hand. When predicting future stock prices, context is irrelevant as one can not yet know the future stock prices (what we are trying to predict). So, we will not be implementing bidirectional RNNs.

Hidden state Size

Furthermore, the dataset presented to us had 4 temporal features to be used in training, excluding Close price for reasons presented in section 2. Using all the features, this sets our hidden state size to 4. Given the difference in nature between Prices and Volume, it is interesting to try to train the model only on prices to see whether accuracy is lost or gained compared to training with volume. We will thus be training:

- **All models with hidden state size of 3** (Without Volume)
- **All models with hidden state size of 4** (With Volume)

Layers

Since Stacked RNNs can be of all variants of RNNs (Stacked GRU,LSTM RNN), we will also be training stacked variants of all these RNNs, with a **2-layer architecture** and comparing the accuracy metric to the unstacked models.

The following table summarizes the models we will be training after hyperparameter adjustment.

5.4. Training

This section focuses on the training process of the models we have defined to be trained in the previous section. We will define metrics to evaluate model proficiency as well as which loss function will be used during training. A baseline training curve will be established, leading to hyperparameter tweaking, which will also be discussed. Finally, the best performing model will be selected as a final model and evaluated on the test set.

Model	Number of layers	Hidden state size
Vanilla	1	3
		4
	2	3
		4
GRU	1	3
		4
	2	3
		4
LSTM	1	3
		4
	2	3
		4

Table 1: Models to be trained after hyperparameter adjustment

5.4.1 Performance Metrics

The task at hand is a regression task, the performance metrics of which are defined below. Consider predictions made by the model notated as $\hat{y}$ and the ground truth notated as y:

Loss Function: Mean Square error

The metric we will be using as a loss function for backpropagation is a commonly used metric in regression tasks, Mean Square error is sensitive to outliers and thus serves will its purpose in tasks where mispredicting outliers is reprehensible (this stock prediction task for instance).

Its expression is given by the following formula:

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2 \tag{5}$$

Where N is the batch size for batch gradient descent training

Root Mean Squared Error (RMSE) Root Mean Squared error is an extension of MSE. The advantage of RMSE compared to MSE is that its units are the same as the prediction and ground truth, which allows for easier interpretation. As its name suggests, its formula is:

$$RMSE(y, \hat{y}) = \frac{1}{\sqrt{N}}\sqrt{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2} \tag{6}$$

Mean Absolute Error (MAE)

Mean absolute error is popular for the same reason RMSE is: it has the same units as the prediction and is easily interpreted as the average difference between predicted

values and ground truth. As such, it is a very visual error metric:

$$MAE(y, \hat{y}) = \frac{1}{N}\sum_{0}^{N}(|y_i - \hat{y}_i|) \tag{7}$$

Mean Absolute percentage Error (MAPE)

An extension of Mean absolute error, MAPE is even more interpreted since it provides a percentage error in case the range of the evaluation set is not known. the error ranges from 1 to 100, providing an easy evaluation metric to use:

$$MAPE(y, \hat{y}) = \frac{1}{N}\sum_{0}^{N}(|\frac{y_i - \hat{y}_i}{y_i}|) \tag{8}$$

Coefficient of determination (R^2) The coefficient of determination is often used in linear regression, as it provides an accurate estimate of the ability of the regression model to predict the variations in the ground truth. It serves the same function for any regression task, and provides a computation of **The proportion of variation in the ground truth that is accurately predicted by the model**.

For the task at hand, for which traders often do not desire the exact value of the stock market, but rather its variations over a given period, this is a very relevant metric, which will be one of our main focuses along with MAPE:

$$R^2 = 1 - \frac{\sum_0^N \left(y_i - \hat{y}_i\right)^2}{\sum_0^N \left(y_i - \bar{y}\right)^2} \tag{9}$$

Where $\bar{y}$ is the mean of the ground truth sample.

5.4.2 baseline training curves

The baseline model for this project will be the ensemble of three models (RNN,GRU,LSTM), trained on a 30-day history set, and evaluated on the test set. The batch size (adjustable later on) originally chosen is $batch_size = 32$. The models are trained for 100 epochs at first. The training curves for all three models are shown below (Fig. 9)

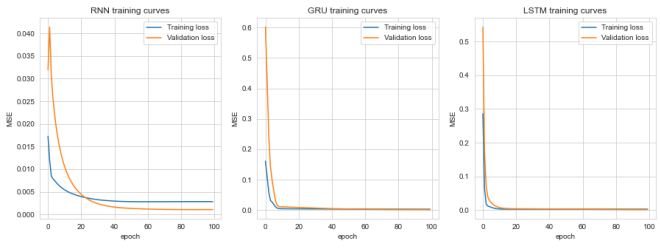

Figure 9: Baseline training curves for 32 batch size, 100 epochs and 30 day history

Furthermore, the predictions made by all three models are shown below in Fig.10 for training (left), test(right) and validation(middle) set:

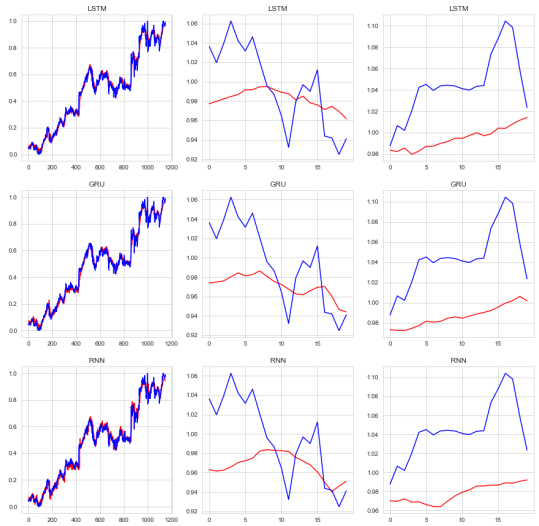

Figure 10: Baseline prediction curves for 32 batch size, 100 epochs and 30 day history

The metrics of these models evaluated on the test set are shown in the following table:

Model	**RMSE**	**MAE**	**MAPE**	R^2
RNN	36	4.1%	33.7	-4.99
GRU	31.5	29.3	3.3%	-3.56
LSTM	28.4	25.4	3.1%	-2.6

Table 2: Metrics for evaluated baseline models , 32 batch size, 100 epochs and 30 day history

We notice that GRU and LSTM perform better than Vanilla RNN, as expected. However, we also notice that all models perform gross underfitting (they do not capture the fine variations of the market, and only capture the general variations, not very well oftentimes. The predictions and mean absolute percentage errors of all three models are shown on the following graph (Fig. 11):

This can also be shown the bad metric values, specifically for R^2 and MAPE. The models are trained again with 1000 epochs to perform a sanity check that this is underfitting, and the metrics are shown in the following table:

As well as predictions for all three models and corresponding APEs on the test set in Fig. 12.

Clearly, increasing the number of training epochs did not help. So, some hyperparameters of model training must be

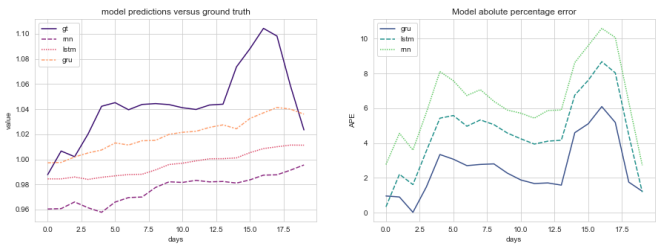

Figure 11: baseline all model predictions and absolute percentage errors

Model	RMSE	MAE	MAPE	R^2
RNN	46	5.5%	45	-9
GRU	32	30	3.6%	-3.7
LSTM	32	31.1	3.7%	-3.7

Table 3: Metrics for evaluated baseline models , 32 batch size, 1000 epochs and 30 day history

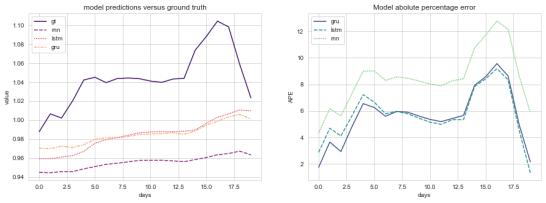

Figure 12: baseline all model predictions and absolute percentage errors for 1000 epochs as sanity check

adjusted, as well as some structural model elements, such as history size foro training. This is adressed in the next section.

5.4.3 Hyperparameter choice

History Size, Batch size

The glaring problem to take care of first if obviously the gross underfitting the model is performing. Trial-and error shows that decreasing batch size and history size leads to less underfitting and more learning fine variations of the stock price. Below are prediction curves for all models for history sizes of **15,7 and 3** days. As well as the associated metrics: (Fig. 13; Table 4)

The ideal History size seems to be 7 days, However we will be testing all combinations for optimal accuracy. Another hyperparameter we wish to trial is batch size, and we will be training all models with 7 day history size for batch sizes of **16,8 and 4**. The results are summarized in Fig. 13 and Table 5:

So, our optimal models seem to be trained With a 7 day

Figure 13: Prediction curves for (a) 15 days, (b) 7 days and (c) 3 days

Model	Days	RMSE	MAE	MAPE	R^2
	15	46	30	5	-9
RNN	7	19	18	2.2	-0.23
	3	39	36	4.7	-2.7
	15	32	45	3.6	-3.7
GRU	7	15	13	1.6	0.08
	3	30	28	3.5	-1.2
	15	32	30	3.6	-3.7
LSTM	7	22	19	2.4	0.28
	3	28	26	3.3	-0.87

Table 4: prediction metrics for varying history sizes

Model	Batch size	RMSE	MAE	MAPE	R^2
	16	23	21	2.6	-.34
RNN	8	16	14	1.7	0.34
	4	9.5	7.3	.91	0.77
	16	24	22	2.7	-.45
GRU	8	10	8	1.1	0.72
	4	8.7	6.6	.83	0.81
	16	20	18	2.2	-.045
LSTM	8	8	7	.9	.79
	4	9.9	7.9	.99	0.75

Table 5: Metrics for varying batch sizes

history and a batch size of 4. However, performing more tests which were not shown here for the sake of length, we find that models trained with history size of 3 and Batch size of 4 perform best between all the previously shown combinations

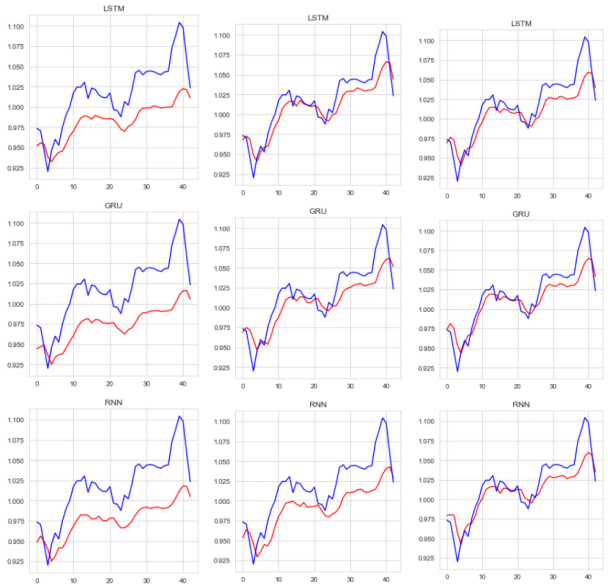

Figure 14: Prediction curves for (a) 16 batch size, (b) 8 batch size and (c) 4 batch size

Layer depth

Now that we have coherent results with history size and batch size, the influence of layer depth remains to be seen. The "optimal" model determined before is trained with 2 layers to determine whether or not accuracy increases or decreases. The results are shown in Fig. 15 and Table 7.

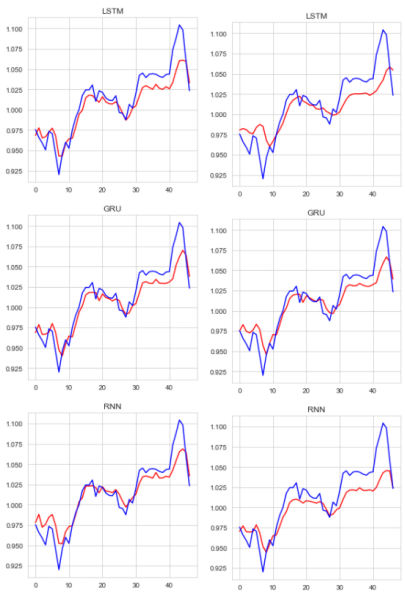

Figure 15: Prediction curves for (a) 1 layer, (b) 2 layers

Accuracy seems to decrease for all models with added layer depth, as training time greatly increases. Furthermore,

Model	layer depth	RMSE	MAE	MAPE	R^2
RNN	2	11.2	8.7	.11	.7
	1	8.5	6.4	0.82	0.83
GRU	2	8.6	8.3	.08	.82
	1	7.9	6.3	0.80	0.81
LSTM	2	11.8	8.1	.105	.66
	1	7.6	5.7	.73	0.86

Table 6: Metrics for varying layer depth

this does not seem to change when adding training epochs as a sanity check. So, we move forward with only one layer.

Hidden state size

Another element to test relating more to model structure is hidden state size, or how many features are taken into account. This stems from the intuition that the feature we are trying to predict is a price, and having our model learn to predict using only prices seems more coherent when trying to predict data of the dame nature.

However, a quick sanity check by removing volume hightens MAPE by about 2%, which is non-negligible and discourages us from using only prices to predict Open price for the following day. Hence, our model's hidden state size will be 4.

5.4.4 final model performance

After adjusting variables of smaller importance such as number of epochs and learning rate, The model which performs best of all the attempted models is One of the three models with the following characteristics:

- **number of epochs:** 100
- **Learning rate:** 0.001
- **History size:** 3 days
- **Number of features:** 4
- **Number of layers:** 1
- **Batch size:** 4

The prediction curves for training (left), test(right) and validation (middle) are shown below (Fig 16), as well as prediction curves with corresponding APEs (Fig. 17), and the final models' prediction metrics on the test set (Table 7).

Figure 16: Predictions for final models

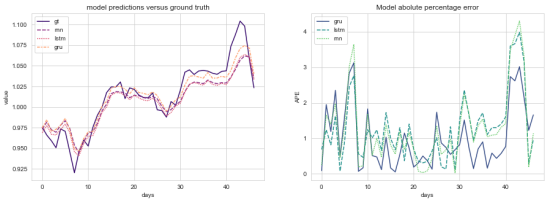

Figure 17: Predictions and APEs for final models

Model	RMSE	MAE	MAPE	R^2
RNN	8.6	8.7	.11	.7
GRU	6.8	8.3	.08	.82
LSTM	8.3	8.1	.105	.66

Table 7: Final models metrics

Given the metrics, Out final choice will be a GRU model with the aforementioned parameters. From a metric standpoint, The MAPE is very good, achieving only 0.6% error, and R^2 shows the model is very good at the given task, predicting 82% of ground truth variations correctly.

List of Tables

List of Figures

References

[1] K. Kohara, T. Ishikawa, Y. Fukuhara, and Y. Nakamura, "Stock price prediction using prior knowledge and neural networks," *Intelligent Systems in Accounting, Finance & Management*, vol. 6, no. 1, pp. 11–22, 1997.

[2] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using lstm, rnn and cnn-sliding window model," in *2017 international conference on advances in computing, communications and informatics (icacci)*, pp. 1643–1647, IEEE, 2017.

[3] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, "Stock price prediction using the arima model," in *2014 UKSim-AMSS 16th*

International Conference on Computer Modelling and Simulation, pp. 106–112, IEEE, 2014.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.