INM427: A Comparative Study in making Stock Price Predictions using CNNs and RNNs

Rohit Sunku rohit.sunku@city.ac.uk

City, University of London 2022/2023

Student ID: 210008043

Abstract: This paper analyses an algorithmic trading dataset found on Kaggle and focuses on the comparison between CNN's, RNNs and LSTMs (a type of Recurrent network) used for predicting stock prices on algorithmic trading data. We compare RNNs (Recurrent Neural Networks) and CNNs (Convolutional Neural Networks), designing their architecture, tuning hyperparameters with our Validation Set and changing hyper-parameters to see how that will affect the validation accuracy. We also analyse LSTMs which is a type of RNN specifically designed for sequential data as a bonus. We use the hyperparameter combinations which give the highest validation accuracy to build our final models. After testing our data, we find that CNNs outperform RNNs in terms of accuracy and computational efficiency in predicting financial time-series data for algorithmic trading.

1. Introduction

The use of artificial neural networks to make stock, cryptocurrency and trading data predictions has been an ever-growing field. Financial Data Scientists and Quantitative Analysts employ a variety of different techniques to forecast future stock prices for brokers and traders to make accurate investments and trades.

The purpose of this paper is to perform a critical analysis of CNNs, RNNs and LSTMs in analysing stock-data. Whilst in industry, a combination of models is usually used along with a fundamental and technical analysis before making informed decisions, we will analyse which of the three models has the highest accuracy pertaining to this dataset.

Reference Analysis

(I) tests LSTMs and CNNs on yahoo stock data, finding that a combination of CNNs for feature extraction combined with LSTMs is the best solution. (II) looks at three datasets and find that CNNs outperform all other machine learning models for time-series forecasting tasks and traditional machine learning models. (III) looks into using stacked autoencoders in combination with LSTM networks for analysis and this outperforms all machine learning models (decision trees, SVR). These models can effectively capture non-linear relationships better than in financial time series data.

Hypotheses Statement

Hypothesis Statement 1: From reference papers highlighted below, each paper utilises a different methodology for predictions, however LSTMs are most used for deep learning, followed by CNNs and then RNNs.

Hypothesis Statement 2: I predict LSTMs will outperform CNNs and RNNs, with RNNs finishing third place with the lowest accuracy.

To do: Analyse references in more detail, add more references in, then finish:

1.1 Convolutional Neural Networks (CNNs)

CNNs are an area of deep learning designed for pattern recognition commonly used for image and video recognition and processing. Inspired by the visual cortex, a convolution is created by overlapping input neuron areas with weight sharing to produce an activation map (kernel) for each filter. A convolution is applied to each channel and summed, with an additional bias giving the output. The set of kernels applied

When a subpart of the data is convolved with a Kernel, it is then reduced by partitioning it into a set of non-overlapping areas, keeping only the neuron which has the highest activation value in each area (Max-

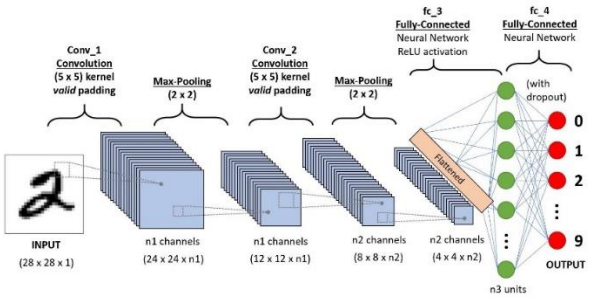
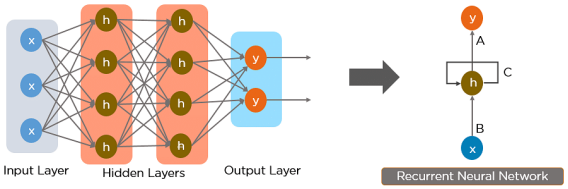


Figure 1: Architecture of a CNN

Pooling). During training, the CNN model is fed with historical stock price data, with the weights of the model being adjusted iteratively to minimise the difference between predicted stock price and true stock price. After the convolutional layers learn complex features of the data and prevent overfitting through dropout layers (interconnected nodes are randomly dropped to prevent overfitting during training) and Max-Pooling, we move onto the fully connected layers. These perform a series of linear transformations on the data, with the ReLU activation function being used to introduce non-linearity to the model. Finally, the fully connected layers connect to an output layer gives the final prediction of the trading price. Once the model is trained, we can use this to predict future trading prices.

1.2 Recurrent Neural Networks (RNNs)

RNNs (Recurrent Neural Networks) are a class of artificial neural networks which are particularly suited for processing sequential data, temporal data and time-series data. The key difference between RNNs and standard ANNs (Artificial Neural Networks) is that the hidden layers in RNNs are designed to maintain a memory of the previous input and use it to process the current input.



RNNs are useful for handling different amounts of input values, as we have seen before RNNs have weights, biases, layers and activation functions; and they also have feedback loop (the feedback loops allow it to take algorithmic trading prices collected over time to make predictions. After scaling the data, we can run this data through an RNN to predict future closing prices. The feedback loop component of the RNN allows for us to enter previous inputs into the input after running the RNN which then converges to a more accurate output.

The drawbacks to RNNs, include they are prone to vanishing/ exploding gradient problems. Furthermore, they only have a limited ability to capture non-linear relationships, and RNNs may struggle to capture complex nonlinear patterns. They are very sensitive to noisy, incomplete data and outliers which can distort final output and potential overfitting. Careful data pre-processing must be completed before initialising this model.

1.3 Long-Short Term Memory (LSTMs)

These are a special type of RNNs containing memory cells and corresponding gate units. Each memory structure has an internal structure allowing for a constant flow of error signals known as the CEC (Constant Error Carousel). This error flow allows the network to bridge long time lags and remember information over extended periods. There are two gate units, the input gate and the output gate controlling the flow of information in and out of the cell. Both these gates are multiplicative, with the input gate protecting the CEC from perturbations of irrelevant inputs while the output gate protects other units from being perturbed. The combination of these two gates regulates the flow of information in and out of memory cells avoiding interference from irrelevant information.

2. Datasets

The data has been sourced from Kaggle, with the largest dataset on NIFTY 100 Indian Stock Data consisting of 1,3,5,10,15,30 and 60 minutes and daily data. We have sourced data from three different companies, Tata Steel, Axis Bank and ICICI Bank data to compare how the models do against different banks.

Each dataset contains an index, date, closing price, highest price, lowest price, open price and volume. The “volume” refers to the total number of shares a particular stock has been traded during the specific period.

A higher volume indicates greater investor interest in the stock resulting in more accurate price discovery. The ‘high’ price is the maximum price within the given time period that buyers were willing to purchase the stock. A larger value suggests a high demand for the stock and this metric provides a good indicator to analysts for the stock’s potential growth and profitability. “Open” refers to the price at which a particular stock started trading at the beginning of a given time period. It is a useful metric for indicating market sentiment and provides insight into how traders view the stock. A higher opening price than the previous days closing price indicates there is positive news/ strong demand for that stock.

Below is an example of one of the datasets we will be using, this is specifically for the ICICI Bank (Time period = Day).

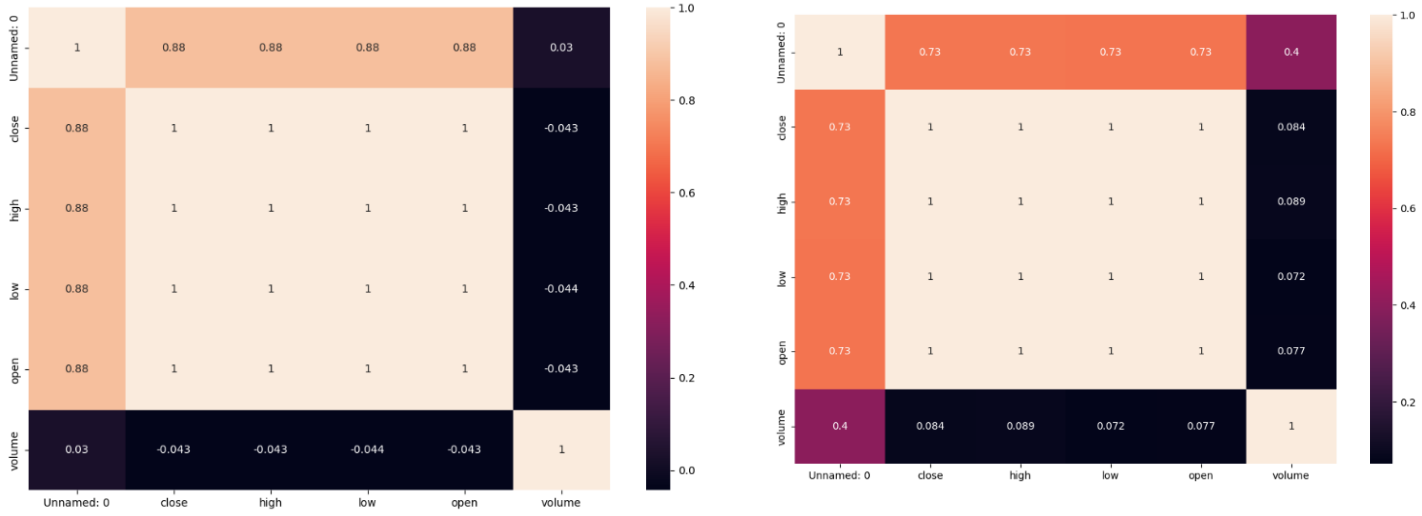
Table 1 – Statistic Summary of The Dataset

Variable	Mean	Standard Deviation	Minimum	Maximum
Date	NA	NA	NA	NA
Close	413.75	188.49	166.36	918.10
High	418.96	189.95	169.05	936.65
Low	408.41	186.74	164.32	915.75
Open	413.82	188.22	167.55	920.00
Volume	1.98 x 10^3	1.45 x 10^7	1.33 x 10^6	2.86 x 10^8

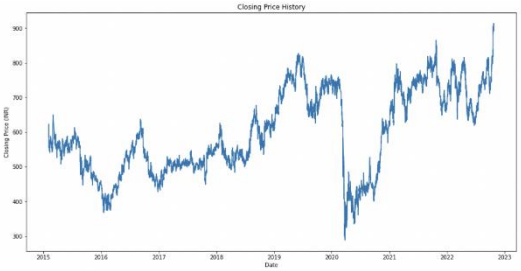
For these algorithmic trading datasets, and stock datasets in general, we can analyse the stock data based on the closing price, and take the high, low and opening prices during this time period. Volume has a very low correlation to the closing price, and any ‘datetime’ data types are discarded for neural networks. We proceed by scaling high, low and open columns before putting through neural networks.

2.1 Initial Data Analysis

From the correlation matrix, we can see that “Unnamed: 0” which is the index of the data, and the volume should be removed. It is important to remove the index since opening, closing, high and low prices will all be different, and the index should not be a factor that is considered training models on stock data. The volume of stocks has a miniscule effect on the closing price, and so it should be discarded.



For both datasets we observe the correlation matrices, the TataSteel day stock data on the right, and the ICICI minute data on the left. Both show a volume has a weak correlation, so we remove this.



This plot shows the closing price history of AXIS. We can see that during 2020/2021 that there was a sudden dip, perhaps due to a company issue. The problem with neural networks is the inability to take into account economic recessions and environmental factors into account, which is the reason why models used by financial data scientists are very complex.

3. Methods

In this section, we detail data cleaning and pre-processing steps and then provide details on training, validation and testing stages. We also delve into the structure of all models, and the architectures and hyperparameters used for them.

3.1 Methodology

3.1.1 Data Pre-Processing

After loading all datasets, we format data types (changing date to date-time for data analysis) and format all datasets to only have open, close, high and low values. All these values are scaled, using a "MinMaxScaler" to ensure all values have a common range between 0 and 1 reducing the impact of difference in scale between features. This will aid our activation function which is sensitive to the scale of input data. We split the data into X being the independent variables (open, high and low prices) and y being the dependent variable (closing price) before applying "train-test-split" twice to get training, validation and testing sets with the corresponding ratio 70:15:15. The methodology used features 15% of the original dataset being used as testing data, for evaluating the models. The other 15% will be the validation set, used for hyperparameter tuning, while the remaining 70% is used as training data.

Moving forward, we convert the 6 data subsets (three for X, three for y) into NumPy arrays and then format them into tensors for input into our model.

We use the Skorch package here, for training and testing our models however we can also use pure Pytorch implementation which would require us to concatenate and convert our 6 subsets into 3 Data Loaders.

3.1.2 Model Selection Process

After designing our models detailed below, we modify the learning rate, number of epochs, momentum, batch size, number of hidden layers for all our models during hyperparameter tuning. We train our models before testing them on our validation dataset. We compare different hyperparameter combinations based on their validation accuracy to find our optimal model for each type.

3.1.3 Algorithmic Comparison Process

Lastly, we compare all three models to see which one performs the best on our testing data. We retrain the best models on our training and validation datasets before doing final testing. We use early stopping criterion to prevent overfitting and improve overall generalisation performance. The model will stop training once the validation loss starts to increase, caused by the model overfitting the training data.

3.2 Architecture and Parameters used for the CNN

The CNN model we use consist of three convolutional layers, followed by three fully connected layers and an output layer. We use the ReLU activation function to transform the summed weighted input from the node and introduce non-linearity to the model, making it capable of learning complex patterns and features within the input data. We choose this function to avoid the vanishing gradient problem which occurs with sigmoid or hyperbolic tangent functions.

CNN Hyper-Parameter Table

Hyper-Parameter	**Value / Name**
Number of Epochs	10 /25/ 50
Batch Size	500 / 1000/ 2000
Learning Rate	0.05 / 0.1/ 0.5
Weight Decay	0.0001
Momentum	0.3 / 1.0 / 3.0
Number of Hidden Layers	[512, 256], [256, 128], [128, 64]
Convolutional Layers Stride	1
Convolutional Layers Kernel Size	3 / 10 / 30
Convolutional Layers Padding	1
Dropout	0.5

Activation Function	ReLU

Note: We change the hyper-parameters highlighted in red.

The number of epochs determines the number of times the entire training dataset will be passed through the network, the batch size takes a sample of the dataset to be trained in each iteration of training, reducing overfitting. Learning rate determines the step size at which models' weights are being updated, with a small learning rate causing the optimization process to take a significant time to converge. A larger learning rate could mean the optimisation process may oscillate around the optimal solution or diverge. We keep the activation function, dropout probability, padding, stride and weight decay the same. We proceed to also experiment with different momentum values, which controls the weight given previous weight updates and updates them based on the average of the previous updates, smoothing them out.

We do not change the stride, padding or use the sliding window method here, as for Stock Data this is not necessary. These hyper-parameters are designed specifically for image processing.

3.3 Architecture and Parameters used for the RNN and LSTM

RNN Hyper-Parameter Table

Hyper-Parameter	**Value / Name**
Number of Epochs	10 /25/50
Batch Size	500 /1000 / 2000
Learning Rate	0.05 / 0.2 /0.5
Momentum	0.3 / 1.0 / 3.0
Number of Hidden Layers	[5,10], [15, 30], [60, 120]
Weight Decay	0.0001
Dropout	0.5
Activation Function	ReLu, LeakyReLU, ELU

Note: We change the hyper-parameters highlighted in red.

We can experiment with different activation functions, perhaps using the LeakyReLU, ELU or Tanh function. Since I have scaled all values between 0 and 1, we can discard the Tanh activation function. I have decided to scale the

We will evaluate the LSTM on the same hyper-parameter combinations as an RNN since we are only this version of RNN as an extension.

4. Results, Findings & Evaluation

4.1 Model Selection

We concluded that the SGD optimiser was preferred in comparison with the Adam and RMSprop optimisers, with RMSprop returning a series of null values during our Pytorch implementation. Furthermore, the optimal loss criterion is MSE loss for this type of problem.

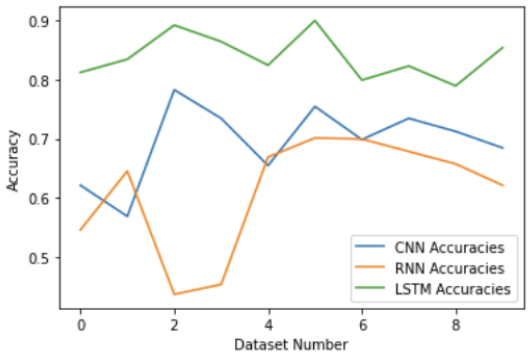
We run our models on our validation data, using the Skorch hyperparameter grid search we get our best hyper-parameters for each model outputted below:

	Learning Rate	Number of Hidden Layers	Number of Epochs	Activation Function	Momentum	Batch Size
RNN	0.5	[5,10]	25	ReLU()	3.0	2000
LSTM	0.5	[15,30]	10	ReLU()	3.0	2000
CNN	0.5	[256, 128]	50	ELU()	1.0	2000

These are the optimal hyper-parameters after running our grid-search on PyTorch. We can see that the preferred learning rate and batch size are at their highest out of the scope due to the size of the datasets. The momentum for RNNs and LSTMs ended up being 3.0, whereas for CNNs the momentum was slightly lower. This is perhaps due to the nature of CNNs where we introduced a large number of pooling layers and a high dropout probability, so a lower momentum was needed to account for overfitting.

4.2 Algorithmic Comparison

We evaluate all our models on MSE score. Sadly, due to an unfortunate error within the matplotlib module, I was unable to output final graphs. I spent two days trying to fix this. But from resources and creating a separate notebook, I was able to do some work on plotting the accuracies of our final models (via guesswork). For 10 different datasets, out of the range of datasets that we have done, we observe results shown below.



We can see that the LSTM plot has the highest accuracy, compared to the RNNs and CNNs due to the nature of the network. LSTMs are a popular choice for analysing algorithmic trading data, and the results clearly show why. The input and output gate feature removes unnecessary data. Also we can observe that RNN accuracy is lower than CNN accuracy for some datasets (1 and 4), which suggests that the type of neural network which is optimal only pertains to a particular dataset and different models will be better for different datasets.

4.3 Comparison to References

(IV) proposes a CNN approach and feature engineering which outperforms traditional machine learning methods. This does not disagree with our results since we have compared CNNs, LSTMs and RNNs. Paper (V) agrees with our results as LSTMs have outperformed the GRU and RNN models in their research.

5. Conclusions, Lessons Learnt and Future Work

From our analysis, we can see LSTMs outperformed RNNs and CNNs agreeing with our initial hypothesis 1 statement. We can see that all three models can predict algorithmic trading data prices accurately. We also see that the initial hypothesis 2 statement was also correct, with CNNs being the least accurate model.

We can learn that models are subjective to their scope of data and different methodologies should be used on different stock data. The specific dataset, feature engineering and hyperparameter tuning are all key factors which make it difficult to definitively say which model is the best. One of the key issues we encountered was working through our

We can experiment with the LeakyReLU and ELU activation functions to see if they are better in analysing the complex patterns within our data and work better with predicting closing prices. We can also do further analysis into other companies on the NIFTY stock exchange, to add more weight to our final model choice. Projects like this can be used to help build machine/deep learning pipelines for financial data science teams to use when being asked to make predictions based on the Stock Exchange.

Other experiments can be done with MLPs and SVM deep learning models. We can also work on using SARIMA and ARIMA models for doing time-series analysis predictions, another way of

6. References

I. "Deep learning for stock prediction using numerical and textual information" by Arjun Sharma and Monika Sharma (2019)
II. "Stock price forecasting using convolutional neural networks" by Shi et al. (2015)
III. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory" by Zhang et al. (2017)
IV. "Stock prediction using convolutional neural networks" by Suresh et al. (2018)
V. "Stock Price Prediction using LSTM, RNN and GRU Neural Network Models" by Althelaya et al. (2020)

A Quick Introduction to PyTorch: Using Deep Learning for Stock Price Prediction | H2O.ai – Also a reference