Predicting Closing Price Movements for Kaggle Competition

Claire Luo **Jerick Shi**

Abstract

1 Introduction

Stock exchanges have very high volatility in which stock prices can change drastically in the final few minutes. At the end of every trading day the official closing prices are established, which serve as important indicators for investors in order to evaluate the performances in certain stocks. In this paper, our goal is to predict closing stock prices for different stocks given data from the order book and the auction book. We will be using data from the Kaggle competition "Optiver Trading at the Close", which consists of data for 200 different stocks for several different time points. Furthermore, we shall be attempting to fit different deep learning models that we have learnt throughout the course in 10-617 to determine whether they achieve higher performances than linear regression models or regular machine learning algorithms. In the end, we discover that machine learning algorithms outperformed linear regression models, whereas linear regression models still performed better than all deep learning models.

1.1 Related Work

Goyal and Welch (2006) (3) analyzed how factors such as interest rates or book-market ratios could be used to predict different stock prices. However, their analysis mainly consisted of using linear model, which in turn, led to their conclusion that "the models are unstable, in that their out-of-sample predictions have performed unexpectedly poorly" and that "the models would not have helped an investor with access only to information available at the time to time the market". In fact, the R^2 of the out-sample predictions would more often be negative, which meant that it is much more preferable to utilize the historical mean rather than a lienar regression model.

Feng (2018) (1) then proposed a new approach, where deep learning can be utilized to predict changes in the market, since these new architectures can capture non-linear patterns in the data. Using neural networks with 2-3 layers with 16-32 nodes, they were able to achieve a positive R^2 of at most 0.02. In the end, Feng states that "deep learning models can rotate input variables and create cutoff hyperplanes. Hence better classification rules" and that "deep learning provides a very fruitful linear of research particularly in empirical asset pricing studies".

On the other hand, Kelly (2019) (2) claims that they "find that “shallow” learning outperforms “deep” learning, which differs from the typical conclusion in other fields such as computer vision or bioinformatics, and is likely due to the comparative dearth of data and low signal-to-noise ratio in asset pricing problems". This is most likely because it is really easy to overfit market data.

Building off of the past research above, we will use our learnings from the deep learning course to determine the usefulness of using deep learning techniques in predicting changes in stock prices, and compare it to current models, such as linear regression and decision trees.

1.2 Background

Our main goal is to continue with Feng's theory (1) where deep learning is a useful tool to analyze nonlinear patterns in the market. Hence, as a baseline model, we fit a linear regression model. As a comparison, we also fit a simple neural network. We then use the mean absolute error (metric) as a measure of predictive accuracy. The linear regression model produced an MAE of 6.33 whereas the neural network produced an MAE of approximately 6.31. However, the neural network produced a negative R^2 whereas the the linear regression model, so its performance is still questionable.

2 Methods

During our midway report, we discovered a few flaws while processing our data. Hence, before fitting the models, we attempt to fix these flaws. While there are several minor flaws including dealing with null data by simply dropping it or splitting it into batches the wrong way, the main error was randomly splitting the data in 80/20. Since we are working with time-series data, this causes look-ahead bias while training. Hence, we now split the the data with the training data being the first 80% of the time periods and the test data being the last 20% of the time periods. Another error was that while evaluating the models, we mainly looked at the MAE and not the R^2 for the models. Hence, in this report, we report both statistics.

Again, we perform our linear regression model as a baseline model, which is the model we are trying to beat. However, in addition to a simple linear regression model, we also utilize Lasso (ℓ_1 regularization), Ridge (ℓ_2 regularization) and ElasticNet (ℓ_1 and ℓ_2 regularization) in order to prevent overfitting.

Our next step is to improve on the neural networks that we performed during the midway report. Whereas we previously only fit one neural network, we fit 8 different neural networks in this report:

Name	Description
DL-64-32-R	FFN with 2 hidden layers (64 and 32 nodes) with Relu Activation
DL-64-32-T	FFN with 2 hidden layers (64 and 32 nodes) with Tanh Activation
DL-128-64-R	FFN with 2 hidden layers (128 and 64 nodes) with Relu Activation
DL-128-64-T	FFN with 2 hidden layers (128 and 64 nodes) with Tanh Activation
DL-256-128-64-R	FFN with 3 hidden layers (256, 128 and 64 nodes) with Relu Activation
DL-256-128-64-T	FFN with 3 hidden layers (256, 128 and 64 nodes) with Tanh Activation
DL-512-256-128-R	FFN with 3 hidden layers (512, 256 and 128 nodes) with Relu Activation
DL-512-256-128-T	FFN with 3 hidden layers (512, 256 and 128 nodes) with Tanh Activation

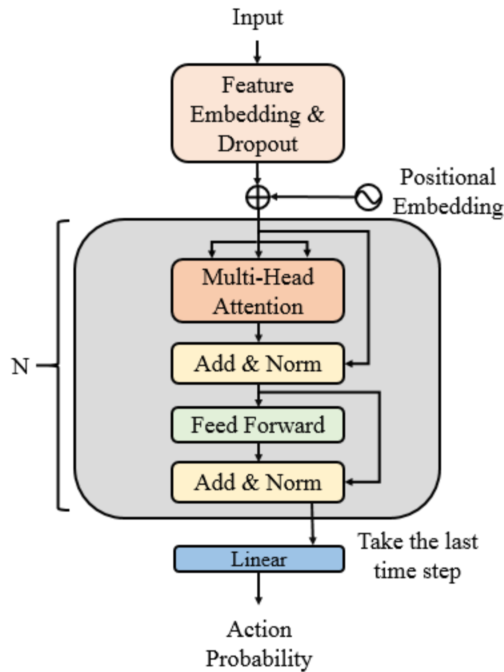
Table 1: List of Feed-Forward Neural Networks utilized

Through experimentation, we realize that the loss converges after around 100 epochs. Hence, all the models are trained on 10 epochs with a learning rate of 0.001 and batch size 128 with the Adam optimizer.

We then consider improving the neural network through 2 different ways: Adding convolution layers or adding LSTM memory cells. For the convolution layers, we apply 1d convolutions to our data, and apply a structure similar to AlexNet (4), but a much smaller scale. Furthermore, our LSTM structure will have 4 LSTM layers with dropout at each layer in order to prevent overfitting. The structures of the model are shown in the Appendix.

We then train the LSTM model in 2 different ways. The first method is directly using all the data to train at each epoch for 10 epochs, and the second method first splits the dataset into each stock, and then trains the model on each stock for 10 epochs. We then summarize our findings in the results section.

After fitting traditional deep learning techniques, we consider more non-traditional techniques taught later in the course. We first utilize the transformer model, which is more optimized by Ntakouris (2021) (5) for time series data. The final structure of the model then looks as follows:



For our transformer model, we have an embedding size of 40, 4 attention heads, 5 transformer blocks, 256 mlp units, and a dropout of 0.1. Due to the large amount of RAM needed to run this code as well as the time needed, this is the only example we could test (since we both have Macs and Google Collab has a limit on the amount of GPU usage per day).

Another model that we use is the autoencoder model, whose architecture comprises of a hidden layer with 64 nodes, a ReLU activation function for encoding, and a decoding layer with the linear activation function. The model is trained on the training dataset for 10 epochs with a batch size of 32. The encoder part of the trained autoencoder is used to extract features from both the training and testing sets, which then serves as inputs for subsequent regression modeling.

We then attempt to improve the performance of the autoencoder model using dropout, in which we introduce a dropout layer after the first dense layer in the encoder. Hence, the final architecture is as follows: An input layer with dimensions matching the number of features, a dense layer with 128 nodes and ReLU activation function, followed by a dropout layer with a dropout rate of 0.2 for regularization, another dense layer with 64 nodes and ReLU activation, and finally, a decoding layer with linear activation.

Lastly, we want to compare our deep learning techniques with other machine learning methods. We train the data on the XGBoost model with the following hyperparameters:

Using the above methods, we are then able to make comparisons amongst the different models and determine whether deep learning methods seems to be more robust in market data than linear models or other machine learning models.

Objective Function	reg:squarederror
Evaluation Metric	mae
Maximum Depth	6
Learning Rate	0.025
Subsample	0.8
Colsample bytree	0.8
Number of Estimators	197
Verbose	True

Table 2: Configuration of XGBoost Model

3 Results

3.1 Baseline Models

3.1.1 Linear Regression Model

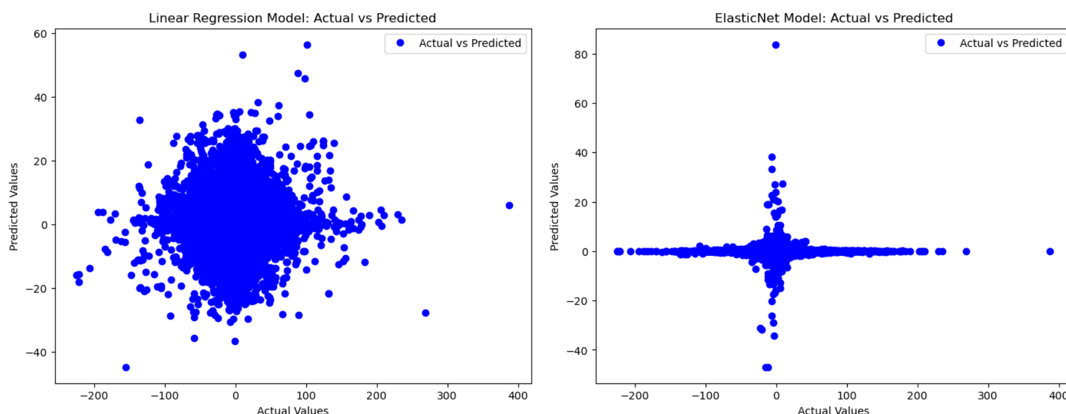
We summarize the results of the different linear regression models in the table below:

	In-Sample MAE	In-Sample R^2	Out-Sample MAE	Out-Sample R^2
Simple Linear Regression	6.398	0.027	6.026	0.007
Lasso Regression	6.491	0.0008	6.059	0.001
Ridge Regression	6.466	0.008	6.045	0.004
ElasticNet	6.492	0.0008	6.059	0.001

Table 3: Performances of Linear Regression Models

Surprisingly, adding regularization seemed to decrease the performance of the linear regression models. This indicates that the data is so noisy that a linear regression model isn't identify all of the important factors within the model, which is reflected in the R^2 values above.

We then analyze how regularization affected the fit of the values in the figures below:



Here, we notice that by adding regularization, the model is more likely to predict 0 for extreme values whereas the original linear regression model seems to have predictions that are much more scattered. Due to this scattering, the simple linear regression seems to have better performance, whereas regularization allowed more predictable patterns.

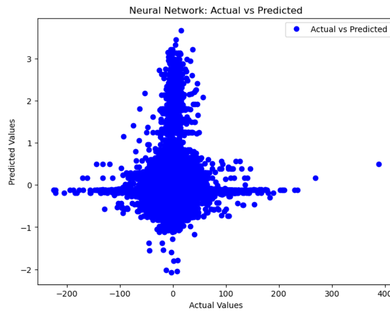
3.1.2 Neural Network Model

We summarize the results of out 8 neural networks in the table below:

	In-Sample MAE	In-Sample R^2	Out-Sample MAE	Out-Sample R^2
DL-64-32-R	6.496	-2.393×10^{-6}	6.062	-0.0007
DL-64-32-T	6.486	0.0030	6.059	0.0012
DL-128-64-R	6.496	-6.973×10^{-7}	6.061	-2.403×10^{-6}
DL-128-64-T	6.495	0.001	6.067	-0.0006
DL-256-128-64-R	6.496	-3.010×10^{-6}	6.061	-1.393×10^{-6}
DL-256-128-64-T	6.490	0.002	6.060	0.0005
DL-512-256-128-R	6.496	-2.642×10^{-6}	6.061	-1.132×10^{-6}
DL-512-256-128-T	6.486	0.003	6.058	0.001

Table 4: Performances of Linear Regression Models

The first observation that comes up is that tanh as the activation functions seems to work better than the relu function, indicating that there are non-linearities in the data. Furthermore, the out-of-sample performs performed the best on the smallest neural network, which is surprising. Looking at the R^2, we see that some of the values are negative, which indicate that neural networks by themselves are not a good fit for the dataset.



By looking at the distributions again between the actual and predicted values for the optimal neural network, it seems to have a similar shape as the ElasticNet model, but with more scattered points around the axis. However, looking tat the y-axis, we realize that all the predicted values are quite small, indicating that the model is highly prone to overfitting. Hence, this might be a reason why direct applications of feed-forward networks do not work optimally.

Overall, the best neural network model does not out-perform the best linear regression model due to overfitting.

3.2 Improved Deep Learning Models

3.3 RNN with LSTM

As mentioned before, we run the LSTM model in 2 different ways, experiment 1 being running all the data at the same time and experiment 2 being partitioning through the different stocks. We summarize the results in the table below:

	In-Sample MAE	In-Sample R^2	Out-Sample MAE	Out-Sample R^2
LSTM 1	6.449	0.013	6.030	0.009
LSTM 2	6.567	-0.012	6.070	-0.002

Table 5: Performances of LSTM Models

Observing the above results, we immediately see that training all of the data at the same time gives much better results. This may be due to all of the stocks sharing some similar pattern which is understood better as a whole, which makes sense, since most stocks follow the market pattern. Furthermore, we notice that LSTM 1 performs much better than all of the previous neural networks,

which makes sense, since we are introducing time-series information, in fact, the out-sample R^2 is around 8 times larger in the LSTM compared to the best feed-forward network.

3.4 Convolutional Neural Network

Similarly to before, we summarize the results in the table below:

	In-Sample MAE	In-Sample R^2	Out-Sample MAE	Out-Sample R^2
CNN	6.449	0.013	6.061	-3.378×10^{-5}

Table 6: Performance of CNN Model

The CNN model seems to do around the same as the neural networks, without much improvement. This is due to the fact that we only have 16 columns. If we were to add more columns, and tune the parameters within the model, we may achieve much better results.

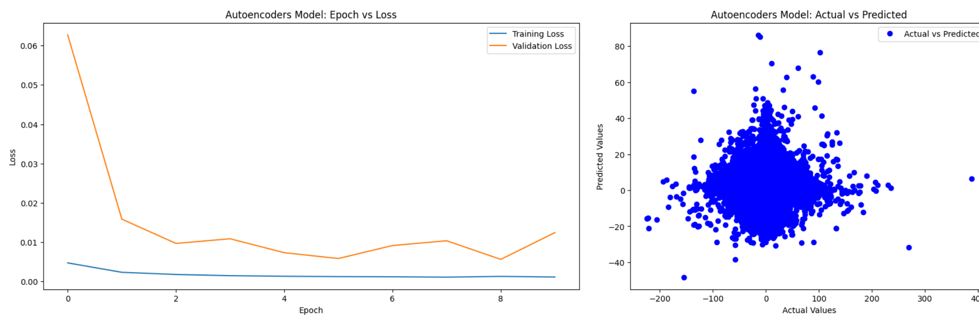
3.5 Other Deep Learning Methods

3.5.1 Transformers

After training, we obtain an out-of-sample MAE of 6.065 and an R^2 of -0.001, which surprisingly performed worse than some of the neural networks. The main reason is most likely due to the noisiness of the data. Transfomers are great and connecting information from different sources, but if the data itself is biased and unclean, it is hard for transformers to make connections. Of course, we could fine-tune the parameters in order to possibly achieve much better performance, but due to the lack of computational resources, such a transformer model is the best we can do for now.

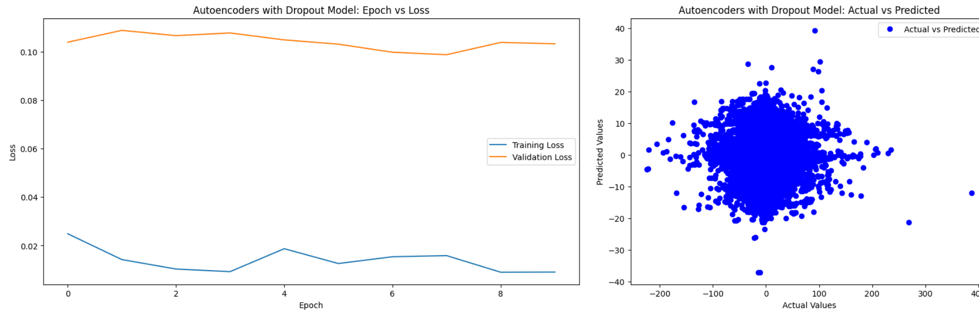
3.5.2 Autoencoders

We summarize the results of training loss and validation loss as follows:

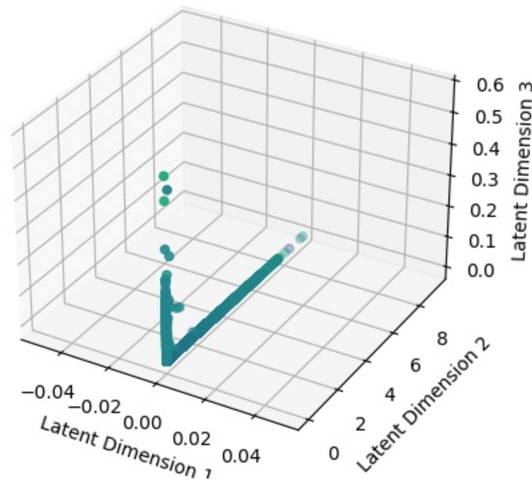


We can observe the validation loss continues decreasing while the training loss remains constant. In this case, the autoencoders model achieved a MAE of 6.4284, providing a quantitative measure of its performance. Overall, the autoencoders model successfully capture underlying patterns and features in the unseen data.

We then add dropout to determine whether there is an improvement:



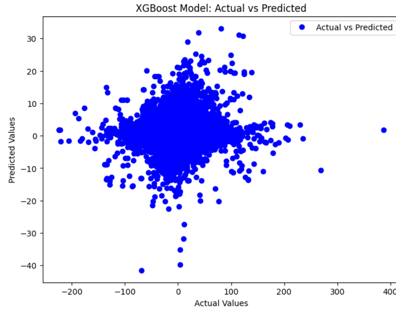
We can observe both the validation loss and the training loss remains constant. In this case, the autoencoders model achieved a MAE of 6.0912, which improves a lot in its performance. However, we notice that the autoencoders model with dropout may encounter a overfitting issue when capturing the patterns and features in the unseen data. We then visualize the latent space below:



As expected from previous models, the model seems to encode most data to a similar value, which is why we see no deviations in one of the dimensions.

3.6 Machine Learning Methods

Using the XGBoost algorithm, we manage to reach a final MAE of approximately 5.9967, which is the best model we found. Looking at the image below, we also see that the model is more likely to make a larger range of predictions compared to previous deep learning methods.



4 Discussion and Analysis

4.1 Final Results

We summarize the results of our best models below:

	Out-Sample MAE
Linear Regression	6.026
Neural Network	6.059
LSTM	6.030
CNN	6.061
Transformer	6.065
Autoencoder	6.091
XGBoost	5.997

Table 7: Final Performances of All Models

Surprisingly, the linear regression model outperformed all of the deep learning methods. This is mainly due to the noisiness of market data, causing it to be difficult to extract non-linear features. However, machine learning techniques seem to perform better than all of the models, which supports Kelly's theory. Within the deep learning methods, the LSTM seemed to perform the best, which isn't a surprise, since we expect this model to work quite well with time series data.

4.2 Limitations

For our models, we did very little cleaning. While we did scale the features, we could improve the data by looking at outliers as well as adding more features such as the average, mean, or standard deviations of the other columns. We also assumed that all the stocks have some similarities, which is why for most of the experiments, we trained on the entire dataset as a whole. Within the models, we did little fine-tuning of the parameters such as those for CNN's or transformers due to the lack of resources and time. Given more time, we can clean the data, add more features, and test more parameters, which will definitely improve the performance, and most likely beat the performance of the linear regression model.

4.3 Conclusion

In this report, we took the dataset from a Kaggle competition in order to predict closing prices of 200 different stocks based off of several different factors. We fit linear regression models, deep learning models, and other machine learning techniques in order to obtain the optimal results. We determined that linear regression still outperforms deep learning models due to the time and resources needed to clean the data and tune the models in order to obtain the optimal performance, and that traditional machine learning techniques are still excellent predictors. Some future approaches include tuning the parameters, cleaning the data, and applying boosting to achieve even better results.

References

[1] Feng, G., He, J. & Polson, N. Deep Learning for Predicting Asset Returns. (arXiv.org,2018,4), https://ideas.repec.org/p/arx/papers/1804.09314.html

[2] Gu, S., Kelly, B. & Xiu, D. Empirical asset pricing via machine learning. *SSRN*. (2018,4), https://papers.ssrn.com/sol3/papers.cfm?abstract id=3159577 paper-citations-widget

[3] Goyal, A. A comprehensive look at the empirical performance of equity premium prediction. (2006,1), https://papers.ssrn.com/sol3/papers.cfm?abstract id=517667

[4] Krizhevsky, H. ImageNet Classification with Deep Convolutional Neural Networks. (2017), https://papers.ssrn.com/sol3/papers.cfm?abstract id=517667

[5] Team, K. Keras Documentation: Timeseries classification with a Transformer model. *Keras*., https://keras.io/examples/timeseries

Appendix

Epoch	Training Loss	Validation Loss
1	0.0047	0.0627
2	0.0023	0.0159
3	0.0018	0.0097
4	0.0015	0.0109
5	0.0013	0.0073
6	0.0012	0.0058
7	0.0012	0.0091
8	0.0011	0.0104
9	0.0013	0.0056
10	0.0011	0.0124

Table 8: Autoencoders Model: Training and Validation Loss for Each Epoch

Epoch	Training Loss	Validation Loss
1	87.5942	87.7211
2	87.0348	87.4299
3	86.6548	87.2265
4	86.5407	86.9997
5	86.4658	87.0504
6	86.3438	87.0130
7	86.2278	87.0092
8	86.1358	87.0818
9	86.0831	86.8331
10	86.2237	86.8090

Table 9: Neural Network Model: Training and Validation Loss for Each Epoch

Epoch	Training Loss	Validation Loss
1	0.0249	0.1040
2	0.0142	0.1089
3	0.0103	0.1067
4	0.0092	0.1078
5	0.0187	0.1049
6	0.0126	0.1032
7	0.0154	0.0998
8	0.0158	0.0988
9	0.0090	0.1039
10	0.0090	0.1033

Table 10: Autoencoders Model with Dropout: Training and Validation Loss for Each Epoch

Table 11: XGBoost Model Results

Epoch	**Test MAE**
1	6.05885
2	6.05631
3	6.05387
4	6.05159
5	6.05048
6	6.04826
7	6.04726
8	6.04539
9	6.04364
10	6.04190
11	6.04032
12	6.03955
13	6.03805
14	6.03657
15	6.03528
16	6.03400
17	6.03280
18	6.03220
19	6.03096
20	6.02983
21	6.02878
22	6.02783
23	6.02687
24	6.02608
25	6.02523
26	6.02448
27	6.02373
28	6.02325
29	6.02271
30	6.02213
⋮	⋮
195	5.99677
196	**5.99671**

Neural Network Architecture:

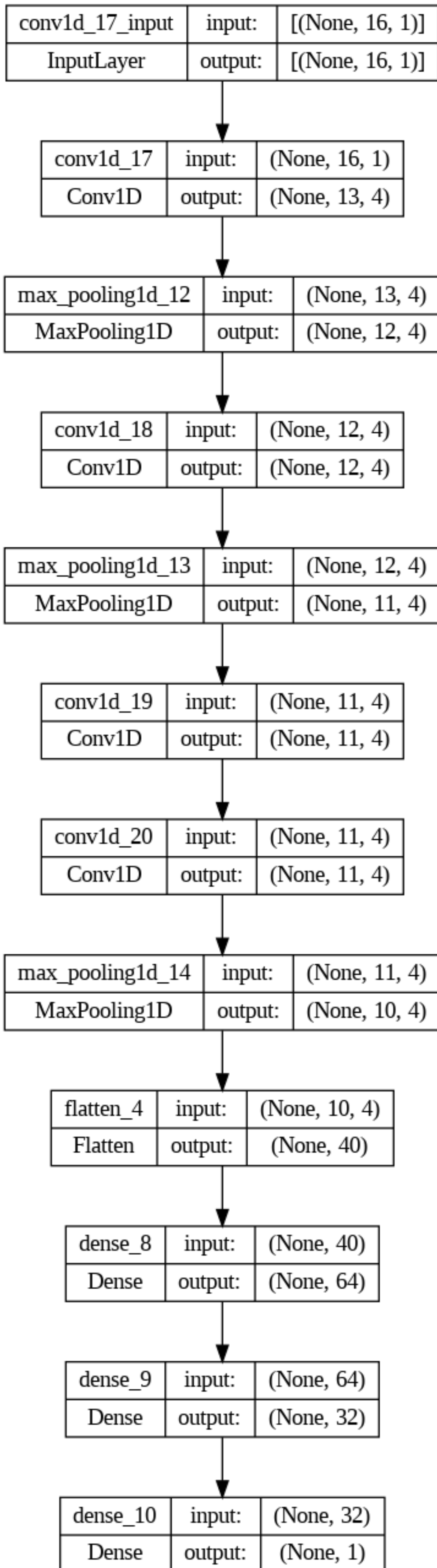
dense_12_input	input:	[(None, 16)]
InputLayer	output:	[(None, 16)]

dense_12	input:	(None, 16)
Dense	output:	(None, 64)

dense_13	input:	(None, 64)
Dense	output:	(None, 32)

dense_14	input:	(None, 32)
Dense	output:	(None, 1)

CNN Architecture:



LSTM Architecture: