A comparison between CNN and LSTM-RNN in stock price prediction

02461 - Introduction to intelligent systems

Alexander Anker Christensen s214614
Erik Buur Christensen s214656
Frederik Winther Bæk s214618

Division of labour:

Alexander: Subsection 2.2, 3.1, 3.4, 3.5, 4.1, 4.2, 5.1, 5.4

Erik: Subsection 2.1, 2.3, 3.3, 3.5, 5.2, 5.3

Frederik: Subsection 1, 2, 2.2, 3.2, 3.4, 3.5, 5.4, 6

DTU Compute
21/01 - 2022

Contents

1 Abstract

The stock market is a very dynamic and volatile system, which is inherently difficult to predict due to external factors such as politics, social media, and unexpected events. There are many tools that investors use in order to predict the market, which includes neural networks. We examine how a convolutional neural network (CNN) compare against a long short-term memory recurrent neural network (LSTM-RNN) at predicting stock prices. We demonstrate, that CNN is able to predict stock prices with a lower loss, whilst LSTM-RNN is better at predicting changes in price trends. Based on this report, we recommend the use of an LSTM-RNN architecture when predicting stock prices using neural networks.

2 Introduction

Neural networks use algorithms to recognize patterns in sets of data, through connections that operate like neurons, which mimics how the human brain functions. Stocks require time series prediction, which means that each set of data is chronological - in the case of stocks, it is ordered by date. Through investigation, CNN and LSTM-RNN are both able facilitate time series prediction. We wish to examine the performance of these two architectures. To do this, we need to establish an understanding of how these network architectures work.

2.1 CNN

In a convolutional neural network there are 3 kinds of layers: The input layer, hidden layers and the output layer. Between each of these layers are weights that are dotted with the input-vector for the layer, to get a new vector that are fed to the next layer. On each of the layers, an activation-function is applied to filter the data.

When training a CNN model it goes through the following loop: Training data is fed, which returns a result. The discrepancy between the predicted result and the actual result is computed with a loss-function. After this, the gradient of the network is computed, which is the loss-function differentiated in relation to each of the weights. This is used to shift the weights slightly in the opposite direction of the gradient. This is repeated for all the remaining training data.

2.2 LSTM-RNN

An LSTM network consists of 4 units, including a cell, an input gate, an output gate, and a forget gate. This allows the network, contrary to a standard feed-forward network, to handle data over an arbitrary time period, which is suitable for making prediction on time series data - in this case understanding patterns in stock market data.

The advantage to LSTM over a regular Recurrent Neural Network is its ability to partially overcome a common problem of "vanishing" or "exploding" gradients. This is caused by long datasets, where the impact of an element becomes less prevalent whilst the model back propagates through the iterations. An LSTM network partially overcomes this by allowing the gradients to stay *unchanged.* What this means is, using the previous states and the internal cells, the network keeps track of what information strikes it as important and what is irrelevant, thereby "remembering" important characteristics.

2.3 Why LSTM-RNN theoretically is better than CNN at predicting stocks

One of the strengths of LSTM networks is the fact that it remembers values that went through its hidden cells, over an interval of time. This makes it capable of changing its weights based on data from both past and current inputs. This should make the network better at working with time-based data, as the network can take data from both fluctuating- and stable periods into account before iterating its weights.

CNN, on the other hand, only deals with isolated cases which makes it more prone to predicting simple patterns as opposed to finding more complex patterns that might emerge if looking at data from a bigger time-scale.

In this experiment, we will attempt to answer the following research question: "How does a CNN network compare to an LSTM-RNN network when predicting stock prices?". We hypothesize that the LSTM-RNN network is more effective at predicting stock prices due to it's ability to spot more complex patterns.

3 Methods

3.1 Stockmarket predictions and variables

When attempting to predict the stock market, it is important to know the various features of the stock data. The data we use comes from so called "candlestick"-graphs. Here, the following 5 features are represented over a time interval:

Open: The marked price of the stock at the start of the interval

Close: The marked price at the end of the interval

High: The highest price a stock has been bought for during the interval

Low: The lowest price a stock has been bought for during the interval

Volume: The amount of times the stock has been traded during the interval

Open interval: The time frame the data has been captured in - in our case a day

These can be used to make qualified predictions about the course of the observed stock. In our case, we use these as the input neurons for both of the networks (Without the open interval, since this just determines the time frame, which in our case is preset).

3.2 Training and testing data

To conduct our experiment, a select few stocks known to be stable. These stocks were picked from the top 10 of the S&P 500 index, which is an index consisting of the largest 500 companies in the United States. The largest company, Apple or AAPL, was picked as the testing stock. The remaining 9 companies in the top 10 will be used to train the CNN and LSTM models. On top of this, the models will also train using data from the VOO index, which is an index that tracks the S&P 500 index.

3.3 Designing the networks

Our networks has input neurons equal to the amount of datapoints it gets fed, for making each prediction. We have 5 datapoints per day of data (as described in 3.1), so the amount of input neurons are 5 times the amount of days we look back on. This amount of days are described in subsection 3.5.

The amount of output neurons we use corresponds to the amount of numbers the network needs to predict. We made 2 of each type of network, one with 1 output neuron, which predicts the closing price the day after the last day of datapoints it was fed, and a network with 2 output neurons, which predicts the high- and low-price of the stock the day after the datapoints it was fed.

For our networks we've chosen to use 2 hidden layers since, through testing, we've found that no more than 2 layers has a significant effect on the outcome. Too many layers might converge on an overfitted model. We've instated a dropout rate in the LSTM-RNN, as described in subsection 3.5, to filter out irrelevant data, while remembering reoccurring important patterns.

We have chosen to use RELU as an activation function in our CNN, which sets all negative values of the input-vector to zero. This filters the data, so low values have little to no impact on the output, while high values shape the model's predictions.

3.4 Hyperparameter tuning

Hyperparameters are variables that define the networks learning process. Once defined, these variables do not change at any time during training. To achieve the most optimal hyperparameters for our CNN and LSTM models, a process of trial and error is necessary. Based on our research, we've found that the epoch value with minimal error without risking overfitting would be in the 500's where we've used 512. The data used for training has been segmented into 21-day sequences, so the network looks back at data from these days, and determines the following day. The sequence length of 21 is used since this is the average length to look back to notice important changes and patterns, which is confirmed by a research paper on a similar experiment [1]. Besides this, a batch size of 64 has been chosen to highlight important reoccurring patterns. To filter out noise, a dropout rate of 0.2 has been instated.

We made an algorithm to find the best learning rate for our models by trying several learning rates, and measuring the loss on the test data for each of these trained models. We used this data to find what learning rate had the lowest average loss. Doing this, we closed in on a good learning rate for both the CNN and the LSTM network. However, we used this algorithm when our networks had one output neuron, and the learning rates we found did not work as well when we implemented the second output neuron. We did not have the time to reproduce the learning rate tuning for our new networks, so we ended up using 0,00001 for CNN, and 0,0025 for LSTM as these values worked alright, but it is obvious that to optimize especially the LSTM model, we would need a better learning rate.

3.5 Measuring accuracy

To measure the accuracy of our models, we have chosen to compute the percentage deviation of each model's predictions of the test-data, from what the correct price was. This is not perfect, as it might not catch if the model always predicts slightly higher or slightly lower, or if the model always predicts the price of the last day of the data it was fed. To get around this, we compute the percentage deviation of always guessing on the day before as a benchmark. This gives us a benchmark to compare our models to when we discuss their actual usability.

In our experiment we have chosen focus on the models with 2 output neurons, that estimate a high- and low price for the coming day. This was chosen since giving an interval where the price would be between in the coming day is more important information than the closing price.

4 Results

4.1 Sample size

In a small pilot example, we found the standard deviation of percentage deviation for CNN and LSTM-RNN to be 0.29 and 7.87 respectively. Based on this, we calculated a sample size of 238 for a 95 % confidence interval with an interval width of 2. This turned out to be too time-consuming, and we ended up using a sample size of 50.

4.2 Benchmark and performance

Our benchmark is, as mentioned previously, predicting the previous day's prices. For this we have calculated a mean percentage deviation of 1.53%. Since the benchmark does not differ, no confidence interval is needed.

Following the experiment, the following results has been observed.

	Mean percentage deviation	**95% confidence interval**
Benchmark	1.53 %	—
CNN	1.47 %	$\pm 3.82 \cdot 10^{-2}\%$
LSTM	9.84 %	$\pm 2.59\%$

We have plotted the predictions of one of each type of network below.

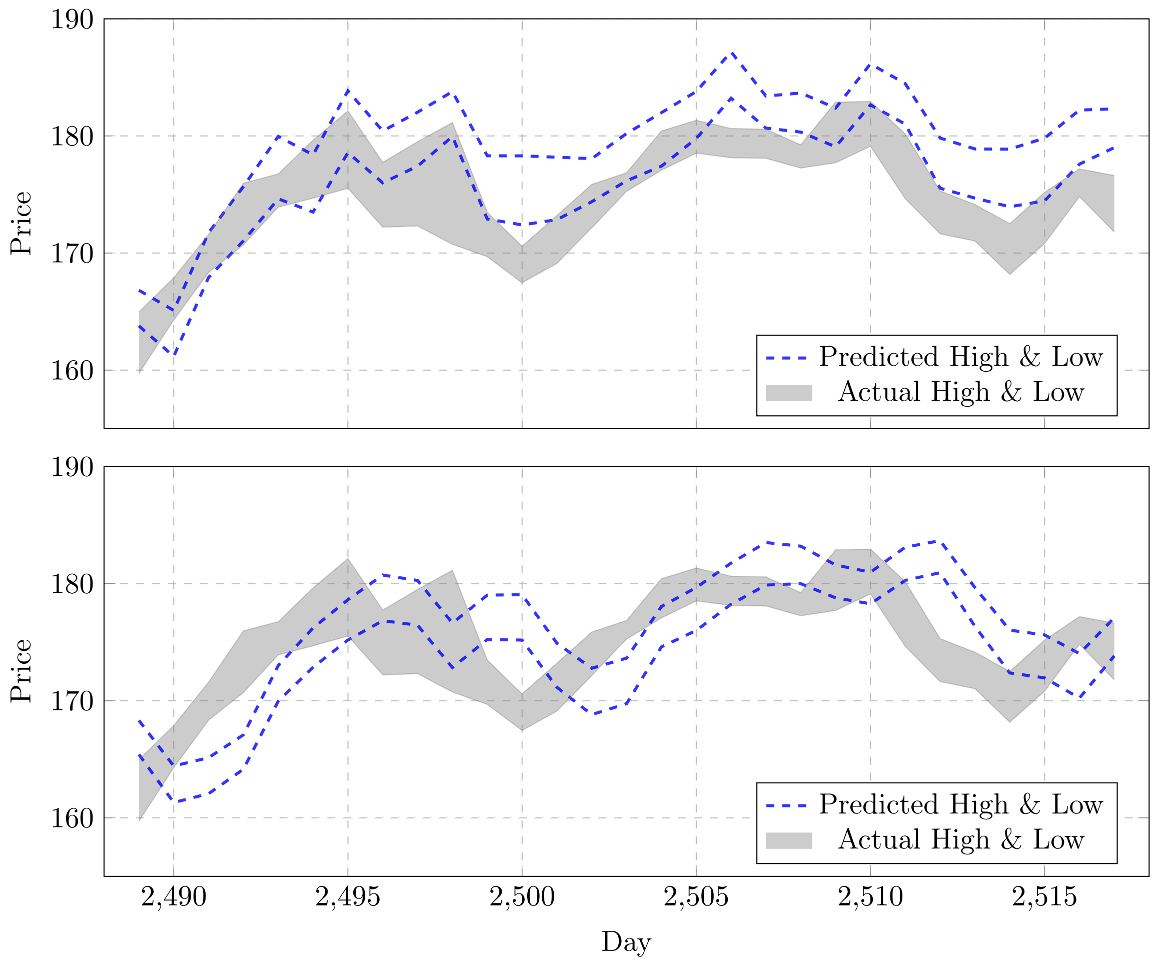

Figure 1: The last 30 days of prediction for the LSTM(top) and the CNN(bottom) network

5 Discussion

5.1 Performance with 1 output neuron

Besides the experiment, we've conducted a similar smaller experiment with only one output neuron, namely the close price of the following day. In this case we hypothesised that the CNN model would be better at noticing an overall trend whilst the LSTM model would be better at actually predicting the following days. This was, to a degree, accurate, yet the accuracy of the CNN was still a lot better than expected. This does make sense, if you take into consideration that we are only looking at one output. We expect it to notice a remarkable difference once multiple outputs comes into play, since the LSTM model allow for many-to-many systems, while the CNN model is limited to many-to-one systems.

5.2 Performance with 2 output neurons

These are the models that we based our experiment on, but contrary to our expectations, the CNN model appeared to be more precise than our LSTM model. While this is unexpected, the reason for this becomes somewhat apparent when looking at the accuracy measurements for individual LSTM models. Because they seem to differ wildly between 2-55%. And through our work in this project, we found out that a big factor of how consistent the training of models was, is setting the correct learning rate, which we were aware could be a problem before running the experiment. After some trial and error, we found that a learning rate of $2.5 \cdot 10^{-4}$ more consistently seemed to make an accurate network, but the percentage error was still around 0.5-2% above the CNN models.

The CNN model seemed to do very well though, as we can fairly confidently say that it is slightly more accurate than our benchmark. It should however be noted, that the margin it performs better than the benchmark is small, and the benchmark is simple enough that it can't reliably be used to make money on the stock market.

5.3 Prediction or optimized pattern

To get an indication of our models performance from a market perspective, we chose to analyse whether the model made good predictions, or only corrected itself based on previous data, and its overall trend. This would arguably be a better indicator of the models quality, as these predictions couldn't be read directly from the previous days' data. Figure 1 in our results show plots of the predictions of one of our CNN models and one of our LSTM models, and compares this to the test data. From these we notice that the CNN network is late when predicting the local extremas, while the LSTM-RNN on multiple occasions predicts these extremas on the same day. Here, it seems as if the CNN network mostly shifts the given data by a day, whilst the LSTM network attempts to predict changes. This would explain the lower accuracy for the LSTM, as it might predict extremas that does not exist, even though it predicts most existing extremas accurately. This is, however still anecdotal evidence, and a more thorough investigation is needed to tell if this is a general rule for LSTM networks.

5.4 Future improvements

To improve both models in the future, we would have to optimize to include external factors such as news articles, or unexpected events. This could be done using sentiment analysis, by reading articles and or social media feeds, and analyze the general mood of the information. Then make a qualified guess as to whether a stock will be impacted by this, and in which way. This information could be implemented as an extra input neuron in both networks. Finally, if we had updated the learning rate finder for the LSTM model, we would most likely have found a more optimal learning rate, which would yield results with lower loss.

6 Conclusion

To answer our question "How does a CNN network compare to an LSTM-RNN network when predicting stock prices?" we conclude that our implementation of CNN is better at getting a lower loss when predicting stock prices when compared to an LSTM-network. However, it would seem LSTM-networks are better at predicting changes in the trends of stock prices. To conclude, if we were to use a neural network to predict stock prices, we would use our implementation of an LSTM-RNN network rather than a CNN network.

7 References

[1] W. Ahmed, M. Bahador, A. Kumar, and Ö. Ekeberg, "The accuracy of the LSTM model for predicting the s&p 500 index and the difference between prediction and backtesting," p. 35,

[2] A. A. Christensen, E. B. Christensen, and F. W. Bæk. "A-CHRI/LSTM-RNN_cnn_comparison: Exam project in the course "introduction to intelligent systems"." (), [Online]. Available: `https://github.com/A-CHRI/LSTM-RNN_CNN_Comparison` (visited on 01/18/2022).

[3] M. Phi. "Illustrated guide to LSTM's and GRU's: A step by step explanation," Medium. (Jun. 28, 2020), [Online]. Available: `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21` (visited on 01/18/2022).

[4] K. Nyuytiymbiy. "Parameters and hyperparameters in machine learning and deep learning," Medium. (Jan. 15, 2022), [Online]. Available: `https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac` (visited on 01/18/2022).

[5] "Neural network definition," Investopedia. (), [Online]. Available: `https://www.investopedia.com/terms/n/neuralnetwork.asp` (visited on 01/19/2022).

8 Appendix

8.1 main_LSTM_HL.py

The code that runs our LSTM-network with 2 output neurons.

```
import numpy as np
import time
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

### Device configuration ###
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

### Hyperparameters ###
features = 5 # Close, Volume, Open, High, Low (Input_size = 5)
seq_len = 21 # length of window
batch_size = 64 # Must be a power of 2
l_rate = 0.0025
n_epoch = 512 # Must be divisible by 8
n_hidden = int((2/3)*(features * seq_len)) # 2/3 input neurons
dropout = 0.2 # Dropout rate

n_input = features * seq_len
n_output = 2

### Training and test files ###
training_files = ["data/AMZN.csv", "data/BRK.csv", "data/FB.csv", "data/GOOG.
    ↪ csv","data/GOOGL.csv","data/JPM.csv","data/MSFT.csv","data/NVDA.csv","
    ↪ data/TSLA.csv", "data/VOO.csv"]
test_file = ["data/AAPL.csv"]

### LSTM Model ###
class LSTM(nn.Module):
    def __init__(self, n_input, n_hidden, n_output, n_layers):
        super(LSTM, self).__init__()
        self.n_hidden = n_hidden
        self.n_layers = n_layers
        self.lstm = nn.LSTM(n_input, n_hidden, n_layers, dropout=dropout,
            ↪ batch_first=True) # n_input -> (batch_size, seq_len, input_size
            ↪ )
        self.linear = nn.Linear(n_hidden * seq_len, n_output) # n_hidden *
            ↪ seq_len -> (batch_size, n_output)


    def forward(self, input):
        # Initialize hidden and cell states
        h_0 = torch.zeros(self.n_layers, input.size(0), self.n_hidden).to(
            ↪ device)
        c_0 = torch.zeros(self.n_layers, input.size(0), self.n_hidden).to(
            ↪ device)

        # LSTM layer
        input = input.view(-1, seq_len, features)
        out, _ = self.lstm(input, (h_0, c_0))

        # Linear layer
        out = out.reshape(len(input), -1)
        out = self.linear(out)
        return out

### Dataset ###
class StockData(Dataset):
    def __init__(self, data_files):
        # Load data
        x_arr = []
        y_arr = []
        for i in data_files:
            data = np.loadtxt(i, delimiter=',', skiprows=1, usecols
                ↪ =(1,2,3,4,5))[::-1]

            for i in range(len(data) - (seq_len + 1)):
                x = data[i:i + seq_len, :]
                y = data[i + seq_len, 3:5]

                # Store data in the dataset
                x_arr.append(x)
                y_arr.append(y)

        # Scale data
        self.scaler = StandardScaler()
        # Flatten data
        x_arr = np.reshape(x_arr, (-1, 5))
        y_arr = np.reshape(y_arr, (-1, 1))

        # Scale data
        data_scaled_x = self.scaler.fit_transform(x_arr)
        data_scaled_y = self.scaler.fit_transform(y_arr)

        # Reshape data
        data_scaled_x = np.reshape(data_scaled_x, (-1, seq_len, features))
        data_scaled_y = np.reshape(data_scaled_y, (-1, n_output))

        # Set the tensors
        self.x = torch.tensor(data_scaled_x, dtype=torch.float).to(device)
        self.y = torch.tensor(data_scaled_y, dtype=torch.float).to(device)

        self.n_samples = self.x.shape[0]

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return self.n_samples

### Print and log ###
def print_and_log(string):
    print(string)
    with open("out/log_LSTM_HL.txt", "a") as f:
        f.write(string + "\n")

if __name__ == '__main__':
    # Clear the log
    with open("out/log_LSTM_HL.txt", "w") as f:
        f.write("")

    # Print the parameter info
    print_and_log(
        "-"*80 + "\n" + f"{'LSTM␣Model␣-␣High␣&␣Low':^80}" + "\n" +
        "-"*80 + "\n" + f"{'Files:':<80}" + "\n" + "-"*80 +
        f"\nTraining␣files:␣\n{training_files}" + "\n"
        f"\nTesting␣files:␣\n{str(test_file)}" +
        "\n" + "-"*80 + "\n" + f"{'Data␣info:':<40}{'Training␣info:':<40}" + "
            ↪ \n" + "-"*80 +
        f"\n{'Features:':<30}{features:<10}{'Learning␣rate:':<30}{l_rate:<10}"
            ↪  +
        f"\n{'Sequence␣Length:':<30}{seq_len:<10}{'Epochs:':<30}{n_epoch:<10}"
            ↪  +
        f"\n{'Batch␣Size:':<30}{batch_size:<10}{'Device:':<30}{'CUDA'␣if␣torch
            ↪ .cuda.is_available()␣else␣'CPU':<10}" +
        f"\n{'Output:':<30}{n_output:<10}{'Hidden␣cells:':<30}{n_hidden:<10}"
            ↪ + "\n" + "-"*80
    )

    # Initialize model
    print_and_log("\nInitializing␣model...")
    timer_start = time.perf_counter()

    model = LSTM(features, n_hidden, n_output, n_layers=2).to(device)
    loss_fn = nn.MSELoss(reduction='sum')
    optimizer = torch.optim.Adam(model.parameters(), lr=l_rate)

    timer_end = time.perf_counter()
    print_and_log('Model␣initialized!␣(' + str(round(timer_end - timer_start,
        ↪  4)) + '␣seconds␣)')

    # Load the training dataset
    print_and_log("\nLoading␣training␣dataset...")
    timer_start = time.perf_counter()

    dataset_train = StockData(training_files)
    dataloader_train = DataLoader(dataset_train, batch_size=batch_size,
        ↪ shuffle=True)

    timer_end = time.perf_counter()
    print_and_log('Training␣dataset␣loaded!␣(' + str(round(timer_end -
        ↪ timer_start, 4)) + '␣seconds␣)')

    # Training loop
    print_and_log("\nTraining...")
    timer_start = time.perf_counter()
```

```
Loss = []

samples = len(dataset_train)
iterations = samples // batch_size
for epoch in range(n_epoch):
    for i, (x, y) in enumerate(dataloader_train):
        x = x.view(-1, seq_len, features)
        y = y.view(-1, n_output)

        # Forward pass
        y_pred = model(x)
        loss = loss_fn(y_pred, y)
        Loss.append(loss.item())

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 25 == 0:
            loss_sci = "{:.4e}".format(loss.item())
            print(f"{f'Epoch␣{epoch+1}/{n_epoch}':^20}␣|␣{f'Step␣{i+1}/{
                ↪ iterations}':^20}␣|␣{f'Loss:␣{loss_sci}':^20}␣\r", end="
                ↪ ")

timer_end = time.perf_counter()
print_and_log('\nTraining␣finished!␣(' + str(round(timer_end -
    ↪ timer_start, 4)) + '␣seconds␣)')

# Load the test dataset
print_and_log("\nLoading␣test␣dataset...")
timer_start = time.perf_counter()

dataset_test = StockData(test_file)
dataloader_test = DataLoader(dataset_test, batch_size=batch_size, shuffle
    ↪ =False)
y_pred_plot = np.array([])

timer_end = time.perf_counter()
print_and_log('Test␣dataset␣loaded!␣(' + str(round(timer_end -
    ↪ timer_start, 4)) + '␣seconds␣)')
    # Test model
    print_and_log("\nTesting...")
    timer_start = time.perf_counter()

    for i, (x, y) in enumerate(dataloader_test):
        x = x.view(-1, seq_len, features)
        y = y.view(-1, n_output)

        # Forward pass
        y_pred = model(x)
        y_pred_plot = np.append(y_pred_plot, y_pred.cpu().detach().numpy())
        loss = loss_fn(y_pred, y)

        if (i+1) % 25 == 0:
            loss_sci = "{:.4e}".format(loss.item())
            print(f"{f'Epoch␣{epoch+1}/{n_epoch}':^20}␣|␣{f'Step␣{i+1}/{
                ↪ iterations}':^20}␣|␣{f'Loss:␣{loss_sci}':^20}␣\r", end="")

    timer_end = time.perf_counter()
    print_and_log('\nTesting␣finished!␣(' + str(round(timer_end - timer_start
        ↪ , 4)) + '␣seconds␣)')

    ### Percentual deviation from target ###
    losspercent = 0
    y_pred_plot = dataset_test.scaler.inverse_transform(y_pred_plot.reshape
        ↪ (-1, n_output))
    plot_data = np.loadtxt(test_file[0], delimiter=',', skiprows=1, usecols
        ↪ =(4,5))[::-1]
    targets = plot_data[seq_len:]
    for i, e in enumerate(y_pred_plot):
        target1 = targets[i, 0]
        target2 = targets[i, 1]
        losspercent = losspercent + abs((e[0]-target1)/target1)
        losspercent = losspercent + abs((e[1]-target1)/target1)
    losspercent = (losspercent/(len(y_pred_plot)*len(y_pred_plot[0])))*100
    print_and_log(f"The␣mean␣percentual␣deviation␣from␣targets␣of␣this␣model␣
        ↪ is␣{losspercent}%")


# Logs the last 30 days of the test dataset
```

```
plot_data = np.loadtxt(test_file[0], delimiter=',', skiprows=1, usecols
    ↪ =(4,5))[::-1]
plot_data_30 = plot_data[-30:]
pred_data_30 = y_pred_plot[-30:]
print_and_log("\nTest␣dataset␣last␣30␣days:" + "\n" + "-"*80)
print_and_log(f"{'Actual␣high':<20}{'Actual␣Low':<20}{'Predicted␣high
    ↪ ':<20}{'Predicted␣low':<20}")
for i, e in enumerate(plot_data_30):
    print_and_log(f"{round(e[0],4):<20}{round(e[1],4):<20}{round(
        ↪ pred_data_30[i][0],4):<20}{round(pred_data_30[i][1],4):<20}")

### Plotting ###
fig = plt.figure(figsize=(16, 9))
sub = fig.subfigures(2, 1)
(top_left, top_right) = sub[0].subplots(1, 2)
bottom = sub[1].subplots(1, 1)

# Set up the loss plot
top_left.plot(Loss, label="Loss␣function")
top_left.legend()
top_left.grid(True)

# Set up the zoomed plot
top_right.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 0], label='
    ↪ High␣&␣Low', color='darkgray')
top_right.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 1], color='
    ↪ darkgray')
top_right.fill_between(np.arange(len(plot_data[:, 0])), plot_data[:, 0],
    ↪ plot_data[:, 1], color='darkgray', alpha=0.2)

top_right.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot
    ↪ [:, 0], label='Predicted␣High', linestyle='--', color='green')
top_right.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot
    ↪ [:, 1], label='Predicted␣Low', linestyle='--', color='crimson')

top_right.axis(
    xmin=len(plot_data[:, 0]) - 30, xmax=len(plot_data[:, 0]),
    ymin=np.min(y_pred_plot[:, 1][-30:]) - 20, ymax=np.max(y_pred_plot[:,
        ↪ 0][-30:]) + 20
    ) # Zoom in on the last 30 days
top_right.legend()
```

```
    top_right.grid(True)

    ### Bottom plot
    # Real data
    bottom.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 0], label='High
        ↪ ␣&␣Low', color='darkgray')
    bottom.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 1], color='
        ↪ darkgray')
    bottom.fill_between(np.arange(len(plot_data[:, 0])), plot_data[:, 0],
        ↪ plot_data[:, 1], color='darkgray', alpha=0.2)

    # Prediction data
    bottom.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot[:,
        ↪ 0], label='Predicted␣High', linestyle='--', color='green')
    bottom.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot[:,
        ↪ 1], label='Predicted␣Low', linestyle='--', color='crimson')

    bottom.legend()
    bottom.grid(True)

    sub[0].suptitle('LSTM␣-␣High␣&␣Low', fontsize=16)
    top_left.set_title('Loss␣function')
    top_right.set_title('Predicted␣closing␣price␣(last␣30␣days)')
    bottom.set_title('Predicted␣closing␣price')

    plt.get_current_fig_manager().window.state('zoomed')
    plt.savefig("out/plot_LSTM_HL.png")
    plt.show()
```

8.2 main_CNN_HL.py

The code that runs our CNN with 2 output neurons.

```
import numpy as np
import time
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

### Device configuration ###
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
### Hyperparameters ###
features = 5 # Close, Volume, Open, High, Low (Input_size = 5)
seq_len = 21 # length of window
batch_size = 64 # Must be a power of 2
l_rate = 0.00001 #3.05e-7
n_epoch = 512 # Must be divisible by 8
n_hidden = int((2/3)*(features * seq_len)) # 2/3 input neurons

n_input = features * seq_len
n_output = 2

### Training and test files ###
training_files = ["data/AMZN.csv", "data/BRK.csv", "data/FB.csv", "data/GOOG.
    ↪ csv","data/GOOGL.csv","data/JPM.csv","data/MSFT.csv","data/NVDA.csv","
    ↪ data/TSLA.csv","data/VOO.csv"]
test_file = ["data/AAPL.csv"]

### CNN Model ###
class CNN(nn.Module):
    def __init__(self, n_input, n_hidden, n_output):
        self.n_input = n_input
        self.n_hidden = n_hidden
        self.n_output = n_output
        super(CNN, self).__init__()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(n_input, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_output),
        )

    def forward(self, x):
        x = x.reshape(len(x), -1)
        out = self.linear_relu_stack(x)
        return out

### Dataset ###
```

```
class StockData(Dataset):
    def __init__(self, data_files):
        # Load data
        x_arr = []
        y_arr = []
        for i in data_files:
            data = np.loadtxt(i, delimiter=',', skiprows=1, usecols
                ↪ =(1,2,3,4,5))[::-1]

            for i in range(len(data) - (seq_len + 1)):
                x = data[i:i + seq_len, :]
                y = data[i + seq_len, 3:5]

                # Store data in the dataset
                x_arr.append(x)
                y_arr.append(y)

        # Scale data
        self.scaler = StandardScaler()

        # Flatten data
        x_arr = np.reshape(x_arr, (-1, 5))
        y_arr = np.reshape(y_arr, (-1, 1))

        # Scale data
        data_scaled_x = self.scaler.fit_transform(x_arr)
        data_scaled_y = self.scaler.fit_transform(y_arr)

        # Reshape data
        data_scaled_x = np.reshape(data_scaled_x, (-1, seq_len, features))
        data_scaled_y = np.reshape(data_scaled_y, (-1, n_output))

        # Set the tensors
        self.x = torch.tensor(data_scaled_x, dtype=torch.float).to(device)
        self.y = torch.tensor(data_scaled_y, dtype=torch.float).to(device)

        self.n_samples = self.x.shape[0]

    def __getitem__(self, index):
        return self.x[index], self.y[index]
    def __len__(self):
        return self.n_samples

### Print and log ###
def print_and_log(string):
    print(string)
    with open("out/log_CNN_HL.txt", "a") as f:
        f.write(string + "\n")

if __name__ == '__main__':
    # Clear the log
    with open("out/log_CNN_HL.txt", "w") as f:
        f.write("")

# Print the parameter info
    print_and_log(
        "-"*80 + "\n" + f"{'CNN Model - High & Low':^80}" + "\n" +
        "-"*80 + "\n" + f"{'Files:':<80}" + "\n" + "-"*80 +
        f"\nTraining files: \n{training_files}" + "\n"
        f"\nTesting files: \n{str(test_file)}" +
        "\n" + "-"*80 + "\n" + f"{'Data info:':<40}{'Training info:':<40}" +
            ↪ "\n" + "-"*80 +
        f"\n{'Features:':<30}{features:<10}{'Learning rate:':<30}{l_rate:<10}"
            ↪  +
        f"\n{'Sequence Length:':<30}{seq_len:<10}{'Epochs:':<30}{n_epoch:<10}"
            ↪  +
        f"\n{'Batch Size:':<30}{batch_size:<10}{'Device:':<30}{'CUDA' if torch
            ↪ .cuda.is_available() else 'CPU':<10}" +
        f"\n{'Output:':<30}{n_output:<10}{'Hidden cells:':<30}{n_hidden:<10}"
            ↪ + "\n" + "-"*80
    )

    # Initialize model
    print_and_log("\nInitializing model...")
    timer_start = time.perf_counter()

    model = CNN(n_input, n_hidden, n_output).to(device)
    loss_fn = nn.MSELoss(reduction='sum')
    optimizer = torch.optim.Adam(model.parameters(), lr=l_rate)

    timer_end = time.perf_counter()
```

```
print_and_log('Model initialized! (' + str(round(timer_end - timer_start,
    ↪  4)) + ' seconds )')

# Load the data training set
print_and_log("\nLoading training dataset...")
timer_start = time.perf_counter()

dataset_train = StockData(training_files)
dataloader_train = DataLoader(dataset_train, batch_size=batch_size,
    ↪ shuffle=True)

timer_end = time.perf_counter()
print_and_log('Training dataset loaded! (' + str(round(timer_end -
    ↪ timer_start, 4)) + ' seconds )')

# Training loop
print_and_log("\nTraining...")
timer_start = time.perf_counter()

Loss = []

samples = len(dataset_train)
iterations = samples // batch_size
for epoch in range(n_epoch):
    for i, (x, y) in enumerate(dataloader_train):
        x = x.view(-1, seq_len, features)
        y = y.view(-1, n_output)

        # Forward pass
        y_pred = model(x)
        loss = loss_fn(y_pred, y)
        Loss.append(loss.item())

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 25 == 0:
            loss_sci = "{:.4e}".format(loss.item())
            print(f"{f'Epoch {epoch+1}/{n_epoch}':^20} | {f'Step {i+1}/{
                ↪ iterations}':^20} | {f'Loss: {loss_sci}':^20} \r", end
                ↪ ="")

timer_end = time.perf_counter()
print_and_log('\nTraining finished! (' + str(round(timer_end -
    ↪ timer_start, 4)) + ' seconds )')

# Load the test dataset
print_and_log("\nLoading test dataset...")
timer_start = time.perf_counter()

dataset_test = StockData(test_file)
dataloader_test = DataLoader(dataset_test, batch_size=batch_size, shuffle
    ↪ =False)
y_pred_plot = np.array([])

timer_end = time.perf_counter()
print_and_log('Test dataset loaded! (' + str(round(timer_end -
    ↪ timer_start, 4)) + ' seconds )')

# Test model
print_and_log("\nTesting...")
timer_start = time.perf_counter()

for i, (x, y) in enumerate(dataloader_test):
    x = x.view(-1, seq_len, features)
    y = y.view(-1, n_output)

    # Forward pass
    y_pred = model(x)
    y_pred_plot = np.append(y_pred_plot, y_pred.cpu().detach().numpy())
    loss = loss_fn(y_pred, y)

    if (i+1) % 25 == 0:
        loss_sci = "{:.4e}".format(loss.item())
        print(f"{f'Epoch {epoch+1}/{n_epoch}':^20} | {f'Step {i+1}/{
            ↪ iterations}':^20} | {f'Loss: {loss_sci}':^20} \r", end="")

timer_end = time.perf_counter()
print_and_log('\nTesting finished! (' + str(round(timer_end - timer_start
    ↪ , 4)) + ' seconds )')
```

```
### Percentual deviation from target ###
losspercent = 0
y_pred_plot = dataset_test.scaler.inverse_transform(y_pred_plot.reshape
    ↪ (-1, n_output))
plot_data = np.loadtxt(test_file[0], delimiter=',', skiprows=1, usecols
    ↪ =(4,5))[::-1]
targets = plot_data[seq_len:]
for i, e in enumerate(y_pred_plot):
    target1 = targets[i, 0]
    target2 = targets[i, 1]
    losspercent = losspercent + abs((e[0]-target1)/target1)
    losspercent = losspercent + abs((e[1]-target1)/target1)
losspercent = (losspercent/(len(y_pred_plot)*len(y_pred_plot[0])))*100
print_and_log(f"\nThe mean percentual deviation from targets of this
    ↪ model is {losspercent}%")

# Logs the last 30 days of the test dataset
plot_data = np.loadtxt(test_file[0], delimiter=',', skiprows=1, usecols
    ↪ =(4,5))[::-1]
plot_data_30 = plot_data[-30:]
pred_data_30 = y_pred_plot[-30:]
print_and_log("\nTest dataset last 30 days:" + "\n" + "-"*80)
print_and_log(f"{'Actual high':<20}{'Actual Low':<20}{'Predicted high
    ↪ ':<20}{'Predicted low':<20}")
for i, e in enumerate(plot_data_30):
    print_and_log(f"{round(e[0],4):<20}{round(e[1],4):<20}{round(
        ↪ pred_data_30[i][0],4):<20}{round(pred_data_30[i][1],4):<20}")


### Plotting ###
fig = plt.figure(figsize=(16, 9))
sub = fig.subfigures(2, 1)
(top_left, top_right) = sub[0].subplots(1, 2)
bottom = sub[1].subplots(1, 1)

plot_data = np.loadtxt(test_file[0], delimiter=',', skiprows=1, usecols
    ↪ =(4,5))[::-1]
# Set up the loss plot
top_left.plot(Loss, label="Loss function")
top_left.legend()
top_left.grid(True)

# Set up the zoomed plot
top_right.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 0], label='
    ↪ High & Low', color='darkgray')
top_right.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 1], color='
    ↪ darkgray')
top_right.fill_between(np.arange(len(plot_data[:, 0])), plot_data[:, 0],
    ↪ plot_data[:, 1], color='darkgray', alpha=0.2)

top_right.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot
    ↪ [:, 0], label='Predicted High', linestyle='--', color='green')
top_right.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot
    ↪ [:, 1], label='Predicted Low', linestyle='--', color='crimson')

top_right.axis(
    xmin=len(plot_data[:, 0]) - 30, xmax=len(plot_data[:, 0]),
    ymin=np.min(y_pred_plot[:, 1][-30:]) - 20, ymax=np.max(y_pred_plot[:,
        ↪ 0][-30:]) + 20
    ) # Zoom in on the last 30 days
top_right.legend()
top_right.grid(True)

### Bottom plot
# Real data
bottom.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 0], label='High
    ↪  & Low', color='darkgray')
bottom.plot(np.arange(len(plot_data[:, 0])), plot_data[:, 1], color='
    ↪ darkgray')
bottom.fill_between(np.arange(len(plot_data[:, 0])), plot_data[:, 0],
    ↪ plot_data[:, 1], color='darkgray', alpha=0.2)

# Prediction data
bottom.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot[:,
    ↪ 0], label='Predicted High', linestyle='--', color='green')
bottom.plot(np.arange(len(y_pred_plot[:, 0])) + seq_len, y_pred_plot[:,
    ↪ 1], label='Predicted Low', linestyle='--', color='crimson')

bottom.legend()
bottom.grid(True)
```

```
sub[0].suptitle('CNN - High & Low price', fontsize=16)
top_left.set_title('Loss function')
top_right.set_title('Predicted closing price (last 30 days)')
bottom.set_title('Predicted closing price')

plt.get_current_fig_manager().window.state('zoomed')
plt.savefig("out/plot_CNN_HL.png")
plt.show()
```