Stock Prediction from Price Charts

Minglei Shi[1]*, Mingyang Fang[2]*, Yiyang Chen[2]*
*equal contribution,
[1]Department of Automation, Tsinghua University
[2]School of Economics and Management, Tsinghua University

1 Introduction

Recently, the financial study field also starts to apply the skill over stock prediction. [Jingwen et al., 2023][1] proposes a method to predict whether a stock will go up or down over the next period by using its price trend images. In this project, we use the model in the paper and our self-constructed network to perform the same task on China's A-share market. We get fancy results which outperform the model constructed in the essay.

2 Implementation

2.1 Project Structure

- Project
    - dataset
    - output
    - plot
    - main.py
    - network.py
    - vit.py
    - engine.py
    - losses.py
    - optimizer.py
    - plot.py
    - cmd.sh

I design my project structure as file trees showed above. Each folder or module is illustrated below.

- **dataset**

  There are A-share 60days dataset in this folder. Supply the data in these experiments.
- **output**

[1] Jiang Jingwen, Kelly Bryan, and Xiu Dacheng. (re-)imag(in)ing price trends. 2023

There is some log information outputted by each experiment such as model information, hyerparameters and so on.

- **plot**

  This folder saves images plotting loss and accuracy curve in these experiments.

- **main.py**

  In main.py, we define the model, criterion, optimizer, training process, plot function and some log information.

- **network.py**

  In network.py, we define three models needed in these experiments, one MLP model and two ConvNets model.

- **vit.py**

  we construct the vision transformer in the experiments. Vit performs better with millions of data, but underperform the CNN model in the small dataset, so, we don't use vit in the experiments. Another main reason is training a vit consume a lot of GPU resources, in the end, the loss outweighs the gain.

- **engine.py**

  In engine.py, we define train_one_epoch function and training process, in the meanwhile, we define validate and test function.

- **losses.py**

  In losses.py, we define different criterion functions include Cross Entropy loss and Euclidean loss.

- **optimizer.py**

  In optimizer.py, we define different optimizer include SGD without momentum, SGD with momentum, Adam, AdamW and other functions.

- **plot.py**

  In plot.py, we define plot functions to plot loss and accuracy curve in our experiments. This module is the same as ploy.py supplied in hw3.

- **cmd.sh**

  The instructions used in these experiments.

2.2 Model

- **CNN1:**

  It is structurally similar to the model in the paper, except for the addition of dilation, which has resulted in the parameters of the fully connected layer being three times that of the original. It contains four convolution layers and a linear projection layer and some activation layers.

  This network has a large number of parameters, which makes it difficult to train, and its performance is not outstanding, so we did not choose it as the baseline.

```
self.conv1 = nn.Sequential(nn.Conv2d(in_channels=in_chans, out_channels=64,
                                     kernel_size=(5, 3), stride=(3, 1), padding=(5+img_H, 2), dilation=(3, 2)),
                           nn.BatchNorm2d(64),
                           nn.LeakyReLU(), nn.MaxPool2d((2, 1)))
self.conv2 = nn.Sequential(nn.Conv2d(in_channels=64, out_channels=128,
                                     kernel_size=(5, 3), stride=(3, 1), padding=(5+img_H//2, 2), dilation=(3, 2)),
                           nn.BatchNorm2d(128),
                           nn.LeakyReLU(), nn.MaxPool2d((2, 1)))
self.conv3 = nn.Sequential(nn.Conv2d(in_channels=128, out_channels=256,
                                     kernel_size=(5, 3), stride=(3, 1), padding=(5+img_H//4, 2), dilation=(3, 2)),
                           nn.BatchNorm2d(256),
                           nn.LeakyReLU(), nn.MaxPool2d((2, 1)))
self.conv4 = nn.Sequential(nn.Conv2d(in_channels=256, out_channels=512,
                                     kernel_size=(5, 3), stride=(3, 1), padding=(5+img_H//8, 2), dilation=(3, 2)),
                           nn.BatchNorm2d(512),
                           nn.LeakyReLU(), nn.MaxPool2d((2, 1)))

self.fc1 = nn.Sequential(nn.Linear(in_features=184320 * 3, out_features=2),
                         nn.Dropout(p=0.5), nn.Softmax(dim=1))
```

Figure 2-1 Structure of CNN1

- **CNN2:**

Contains four convolution layers and a linear projection layer and some activation layers.

This network framework is a reproduction of the network described in the paper, and its structure will not be reiterated here. This is our baseline model.

This network performs quite well, with the accuracy of the best model reaching 57.13%, demonstrating the success of the framework built by the original authors of the paper and its strong generality.

```
self.layer1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=(5,3), stride=(3,1), padding=(8,1)),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(negative_slope=0.01, inplace=True), nn.MaxPool2d((2, 1), stride=(2, 1)),
)
self.layer2 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=(5,3), stride=(3,1), padding=(8,1)),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(negative_slope=0.01, inplace=True), nn.MaxPool2d((2, 1), stride=(2, 1)),
)
self.layer3 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size=(5,3), stride=(3,1),padding=(8,1)),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(negative_slope=0.01, inplace=True), nn.MaxPool2d((2, 1), stride=(2, 1)),
)
self.layer4 = nn.Sequential(
    nn.Conv2d(256, 512, kernel_size=(5,3), stride=(3,1), padding=(8,1)),
    nn.BatchNorm2d(512),
    nn.LeakyReLU(negative_slope=0.01, inplace=True), nn.MaxPool2d((2, 1), stride=(2, 1)),
)
self.fc1 = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(184320, 2),
)
self.softmax = nn.Softmax(dim=1)
```

Figure 2-2 Structure of CNN2

- **CNN3:**

Contains three convolution layers and two linear projection layers and some activation layers.

The main difference between CNN3 and CNN2 lies in the depth and number of channels of the network. Moreover, the original model in the paper tended to focus on the vertical dimension. In CNN3, we do not distinguish between the horizontal and vertical dimensions.

Despite its smaller size, this model demonstrates impressive potential. It even outperformed our larger baseline model, with an accuracy of 57.18%. It achieves a perfect balance between training time and model performance. It is worth mentioning that if the model is further reduced in size, the performance will decrease significantly. This model is the one that shows the best marginal effect of increasing the size of the model.

```
self.conv1 = nn.Conv2d(in_channels=3,out_channels=16,kernel_size=3,stride=1,padding=0)
self.conv2 = nn.Conv2d(in_channels=16,out_channels=32,kernel_size=5,stride=1,padding=0)
self.conv3 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=5,stride=1,padding=0)
self.dropout1 = nn.Dropout(0.25)
self.fc1 = nn.Linear(52480, 512)
self.fc2 = nn.Linear(512, 2)
```

Figure 2-3 Structure of CNN3

- **LSTM**

The LSTM model is an additional model we construct compared to CNN, and it proves that LSTM has a better performance than CNN when deal with sequence data in this assignment. Because the number of features is relatively small (only 5 features with open, close, high, low and volume), the input size of LSTM is 32 and the hidden size in LSTM is 64 (larger hidden size with 128 can reduce the performance). And we set 3 layers to train the data (less layer will reduce the accuracy and more layers won't be more accurate but only reduce the efficiency). The LSTM has better performance compared to CNN model with test accuracy **63.47%**.

```
self.linear = nn.Linear(input_size, 32)
self.lstm = nn.LSTM(32, hidden_size, num_layers, batch_first=True)
self.relu = nn.ReLU()
self.fc = nn.Linear(hidden_size, output_size)
```

Figure 2-4 Structure of LSTM

2. 3 Train & Test

We use instructions in cmd.sh and use the control variables method to conduct training and testing experiments.

3 Experiments

Description: Since the hyerparameters of each network is infinite, we select a few that are representative to illustrate the results.

3. 1 Hyerparameters and Results

Table 3-1 Model performance with different hyerparameters

Tag	Model	Epochs	Learning rate	Weight decay	Precision	Recall/ Sensitivity	Specificity	Accuracy
A.1.1	CNN1	2	1e-05	1e-04	Nan	0.00%	100.00%	48.54%
A.2.1	CNN2	2	1e-05	1e-04	59.09%	45.85%	66.35%	55.80%
A.2.2	CNN2	20	1e-05	0	59.20%	52.12%	61.92%	56.88%
A.2.3	CNN2	20	1e-04	1e-05	51.46%	100.00%	0.00%	51.46%
A.2.4	CNN2	20	1e-05	1e-04	58.95%	54.95%	59.44%	57.13%
A.2.5	CNN2	20	5e-05	1e-04	51.46%	100.00%	0.00%	51.46%
A.2.6	CNN2	20	1e-04	1e-04	51.46%	100.00%	0.00%	51.46%
A.2.7	CNN2	20	5e-04	1e-04	Nan	0.00%	100.00%	48.54%
A.2.8	CNN2	20	1e-03	1e-04	Nan	0.00%	100.00%	48.54%
A.3.1	CNN3	1	1e-05	1e-04	55.68%	28.05%	76.32%	51.48%
A.3.2	CNN3	5	1e-05	1e-04	57.06%	44.97%	64.12%	54.26%
A.3.3	CNN3	20	1e-05	0	59.05%	54.77%	59.74%	57.18%
A.3.4	CNN3	20	1e-05	1e-04	58.60%	54.51%	59.18%	56.77%

A.3.5	CNN3	20	5e-05	1e-04	57.62%	51.73%	59.66%	55.58%
A.3.6	CNN3	20	1e-04	1e-04	56.58%	56.62%	53.93%	55.31%
A.3.7	CNN3	20	5e-04	1e-04	59.28%	47.12%	65.69%	56.13%
A.3.8	CNN3	20	1e-03	1e-04	Nan	0.00%	100.00%	48.54%
B.1.1	LSTM	20	1e-03	0	59.93%	61.24%	65.40%	63.49%

3. 2 Plot

- **CNN**

  First, we demonstrate a typical example of failed training. The model fails to converge.

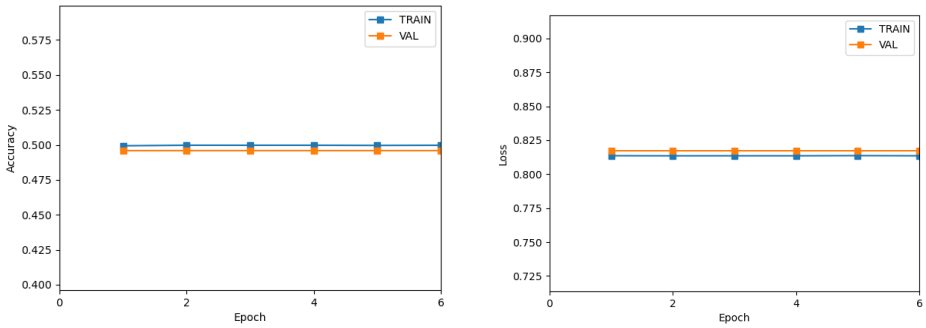

Figure 3-1 Loss and Accuracy A.2.5

The following two groups of figures show the loss and accuracy curves of two well-performing models.

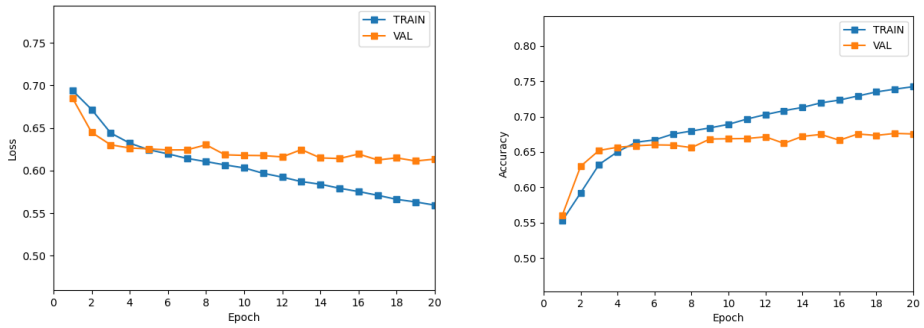

Figure 3-2 Loss and Accuracy A.2.4

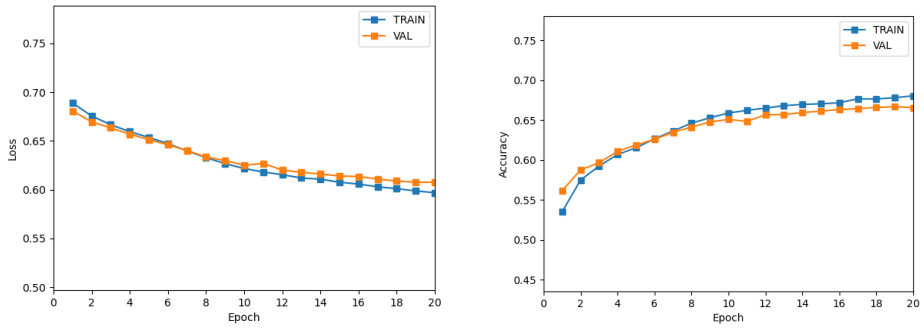

Figure 3-3 Loss and Accuracy A.3.3

Here are the confusion matrices for the two best performing models. We can see that they share a common feature, which is that their precision is relatively high, and their recall is relatively lower. This means that for the stocks you predict to rise, there is indeed a relatively high probability of them rising, but you may not cover too many of the stocks that will rise. In stock return prediction, this is actually a good phenomenon because if funds are limited, we hope to make more accurate investments.

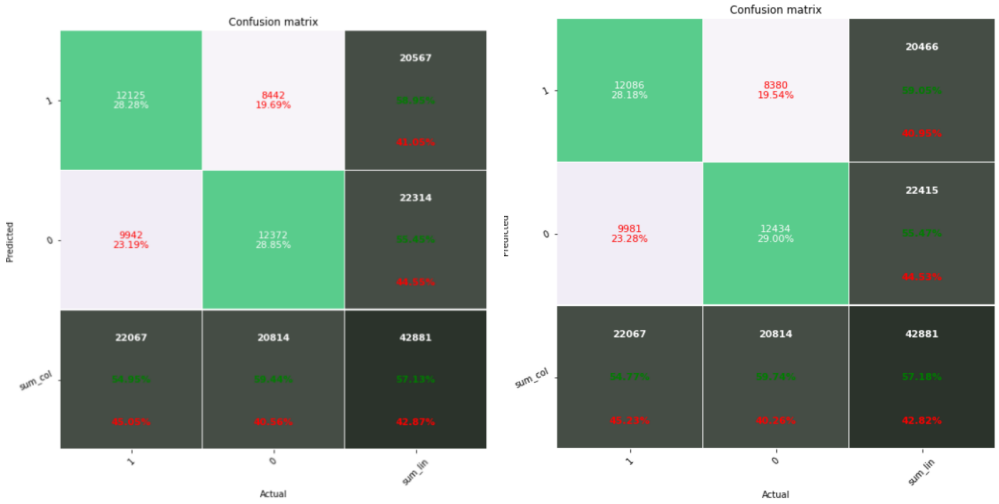

Figure 3-4 Confusion Matrix A.2.4 & A3.3

- **LSTM**

The following two groups of figures show the loss and accuracy curves of LSTM model.

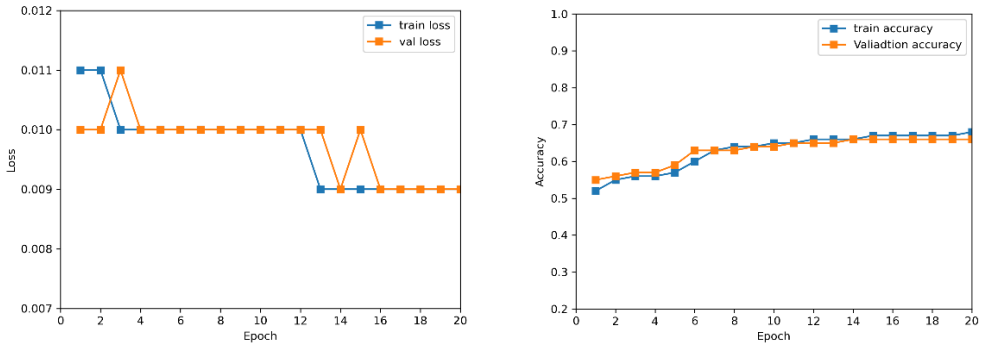

Figure 3-5 Loss and Accuracy of LSTM

Here are the confusion matrices for the LSTM model.

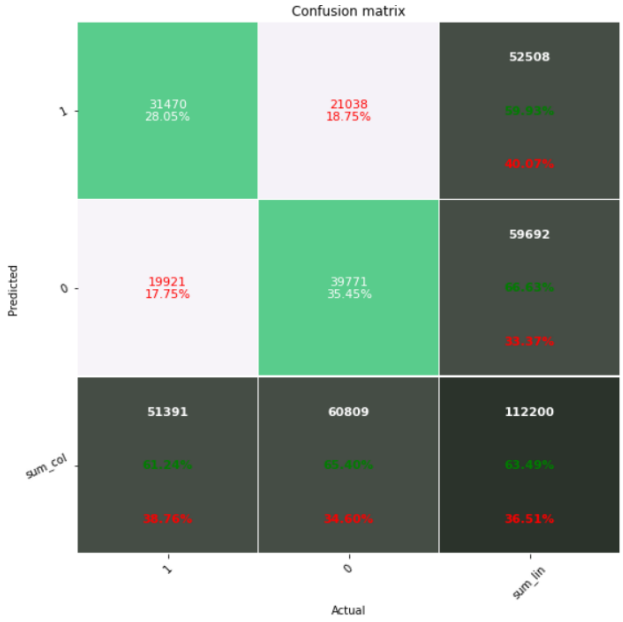

Figure 3-6 Confusion Matrix of LSTM

3. 3 Analysis

Fixed Parameters

- **Batch Size**

We know that batch size only has slight influence on our experiment so we keep the batch size fixed at 96.

- **Weight decay**

In our experiment, we set two kinds of weight decay 0 and 0.0001, and mainly 0.0001. Since the dataset is small and model has limited parameters, weight decay term doesn't play a key role in our accuracy.

- **Criterion**

Since we are solving a classification problem, so we fix the criterion as cross entropy loss.

Variable Parameters and Analysis

- **Dataset**

Based on the A-share market database, we obtained better results than the original paper, indicating that the A-share market may be more inefficient compared to the US stock market and offers more potential investment opportunities that await discovery through superior trading strategies and more complex models such as neural networks.

  - **Dataset of LSTM**

In LSTM model, we construct dataset by ourself using csv. file. First, we use data before 20150830 as the train set (and then split to train and validation set with size 7:3) and data after 20150830 as the test set both with the batch size = 128. And then in each company, we randomly choose 300 sequences (helps to reduce the data redundancy) which include price and volume information of 60 days as the input and label is the 5-day ahead return after the 60 days. And we

also standardize the inputs of our train, validation and test set.

- **Max Epoch**

In the experiments, we adopted an early stopping strategy to select best models (the maximum epoch is 20). If the average validation loss doesn't decrease for 5 consecutive epochs, we stop training. It was observed that when the learning rate is low, accuracy converges over a longer period so that it's harder to trigger the early stopping and the models end up using 20 epochs.

- **Learning Rate**

In the experiments, we set 0.001, 0.0005, 0.0001, 0.00005, 0.00001 in different experiments.

We can find that the models perform well with small learning rate i.e., 0.00001 and underperform with larger learning rate. We noticed that when the learning rate is too high, the training is likely to fail, i.e., the model will always output 1 or always output 0, as shown in the results of A1.1, A.2.7, A.2.8 and A.3.8 in Table 2-1. The reason may be that when the learning rate is too high, the model cannot accurately capture the features of the data, so it ended up choosing the laziest approach of guessing all 1s or 0s.

In the experiments of LSTM, we set learning rate at 0.001 and weight decay at 0 (almost do not have overfitting problem) to achieve the best performance of our model.

- **Optimizer**

At the beginning of the experiment, we tried both SGD and Adam as optimizers and found that Adam usually performs better than SGD. Therefore, we uniformly chose Adam as the optimizer in the final experiments.

4 Conclusion

In our experiments, we applied different kinds of algorithms: CNN and LSTM to predict the stock price using the historical price and volume information. The main conclusion is that when dealing with the longer sequence (60 in this case), LSTM has a better performance than CNN (this also may because we have more data for training LSTM model). The other important thing is the fact that not the deeper the natural network is, the higher its accuracy is, when the features of data is relatively small, network with less layers (both in CNN and LSTM) can capture all the information it needs with higher efficiency.

5 Acknowledgment

Thanks for the homework! It's nice to practice what I have learned, especially the implementation of CNN and LSTM models using pytorch. And I have learned how to construct a project using structured file organization. Thanks for checking! Best wishes.