Project1

Tian Tian

2021011048

Task 1

1. Data Splitting

The loaded dataset is split into training, validation (80%-20% split), and test sets. The data is converted into batch data loaders (`train_loader`, `val_loader`, and `test_loader`) using `torch.utils.data.DataLoader`.

2. Model Structure

A CNN model is defined, which includes two convolutional layers (`self.conv1` and `self.conv2`), a max-pooling layer (`self.pool`), two fully connected layers (`self.fc1` and `self.fc2`), and a Sigmoid activation function.

The model's parameters are updated using either the Adam optimizer or the SGD optimizer, and binary cross-entropy loss (BCELoss) is used to measure the difference between the model's output and the true labels in a binary classification task.

Our final model structure is:

```
CNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=69120, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

3. Hyperparameter and Model Structure Adjustment

Adjustment	batch	epoch	learning_rate	accuracy
Initial	64	5	0.01	0.5021
learning_rate	64	5	0.001	0.5290
epoch	64	10	0.01	0.5003
learning_rate	64	10	0.001	0.5569

During iterations, we found that a smaller learning rate (lr) optimizes more correctly without falling into "traps".

The iteration curve shows that within 10 epochs, the curve gradually converges as the number of iterations increases.

Using batch = 64, epoch = 10, lr = 0.001, we continue to adjust the model structure:

Conv1,2	pool	fc1,2	accuracy
3,32,3—32,64,3	2,2	64x45x24, 128—128,1	0.5569
3,64,3—64,128,3	2,2	128x45x24, 256—256,1	0.5412
3,32,3—32,64,3	2,1	64x45x24, 128—128,1	0.5218
3,64,3—64,128,3	2,1	128x45x24, 256—256,1	0.5569

From the perspective of convolutional layers, the kernel size determines the receptive field of each kernel on the input image. Smaller kernels are used to capture local features, while larger kernels capture larger structural features. More kernels capture more different types of features.

From the perspective of pooling layers, max-pooling retains the most significant features, and average-pooling averages the features. The kernel size of the pooling layer determines the window size of the pooling operation, controlling the size of the output feature map. Larger pooling kernels reduce the spatial dimension of the feature map, thus reducing model complexity.

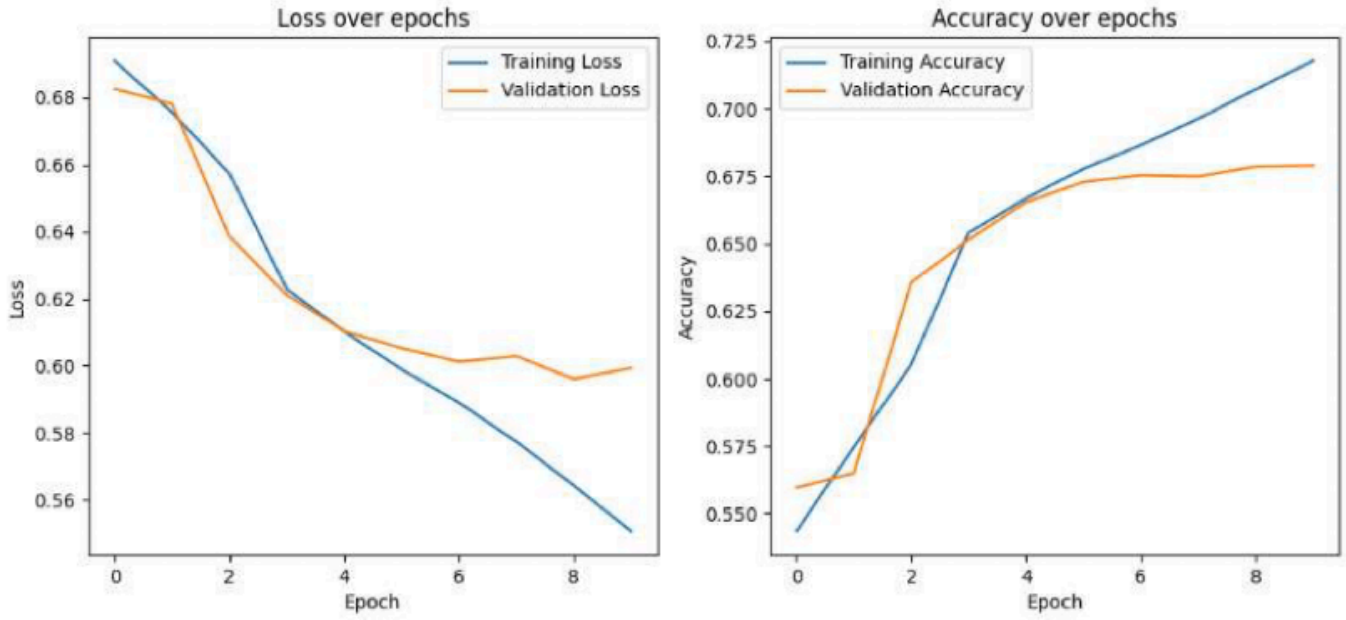
From the perspective of fully connected layers, the number of hidden units controls the output dimension of each fully connected layer, affecting the model's expressive ability. More hidden units can capture more complex feature relationships but may lead to overfitting.

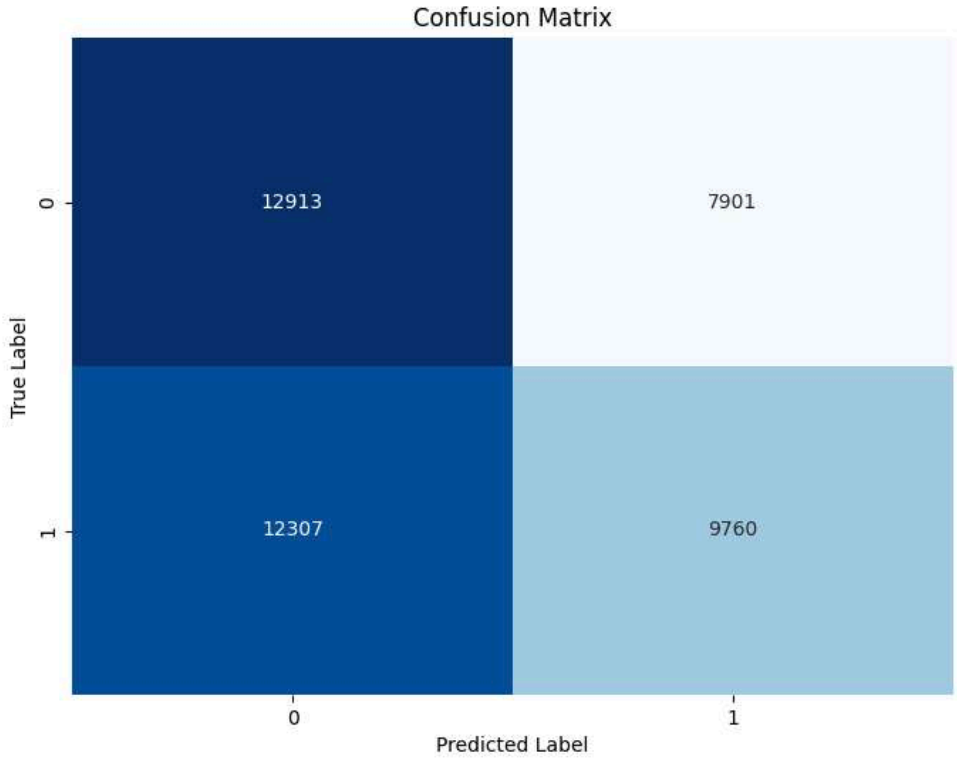
Using conv1,2 = 3,32,3—32,64,3; pool = 2,2; fc1,2 = 64x45x24, 128—128,1, we continue to adjust:

Optimizer	Loss	Accuracy
Adam	0.5993	0.5569
SGD	0.6218	0.5361

4. Results Analysis

We plot the loss and accuracy curves for Adam optimizer, BCELoss loss function, batch = 64, epoch = 10, lr = 0.001, conv1,2 = 3,32,3—32,64,3, pool = 2,2, fc1,2 = 64x45x24, 128—128, 1 as shown below:





By observing the confusion matrix, we found that the matrix has high diagonal values, indicating that our CNN model performs well on that category.

$$Precision = \frac{TP}{TP + FP} = \frac{9760}{9760 + 7901} = 55.26\%$$

$$Recall = \frac{TP}{TP + FN} = \frac{9760}{9760 + 12307} = 44.23\%$$

Task 2

We attempted various models including LSTM, GRU, and standard RNN with different hyperparameters. This section documents our attempts, thoughts, and reflections.

1. Data Splitting

According to the project requirements, we used data from 2010 to 2012 as the training set and data from 2013 as the test set, without using a validation set. We used the past 22 days of data for each trading day to predict the price change direction for the next five days.

Considering the data is time-series data and we use past data to predict future changes, using cross-validation can easily introduce future data into the training, leading to severe overfitting. Therefore, we directly followed the requirements for data splitting and preprocessing.

We initially used the MinMaxScaler normalization function to further normalize the preprocessed data provided in the course. However, we found that the prediction results of the model turned to all zeros after training. Normal training was only achieved by normalizing once. This suggests that if data has already been normalized, repeated normalization might obscure the relative sizes of different features, causing the model to lose the characteristics of the original data during training. The double normalized data's range became too small, causing all data to approach zero, leading to ineffective learning by the model. Additionally, this could result in the gradient vanishing problem, as the data compressed to a very small range, making gradients very small during updates and preventing effective parameter adjustments, leading to constant zero outputs from the model.

2. Standard RNN

(1) Model Structure Introduction

The RNN model is designed to handle sequential data through its memory function, making it effective for time-series data.

Our final RNN model structure is as follows:

```
EnhancedRNNClassifier(
  (rnn1): RNN(6, 80, batch_first=True, dropout=0.1)
  (rnn2): RNN(80, 80, batch_first=True, dropout=0.1)
  (rnn3): RNN(80, 80, batch_first=True, dropout=0.1)
  (batch_norm): BatchNorm1d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc1): Linear(in_features=80, out_features=80, bias=True)
  (fc2): Linear(in_features=80, out_features=2, bias=True)
  (attention): Linear(in_features=80, out_features=1, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

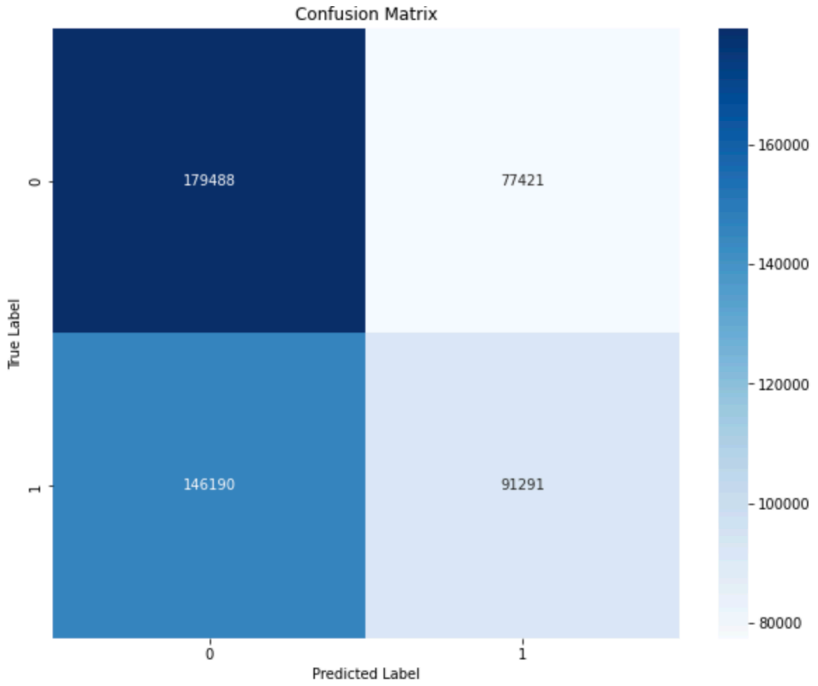
The final hyperparameters we used are as follows:

```
# Define model hyperparameters
hidden_size = 80
num_layers = 3
num_epochs = 5
dropout_rate = 0.1  # Set dropout rate
learning_rate = 0.00005  # Set learning rate
weight_decay = 0.01
```

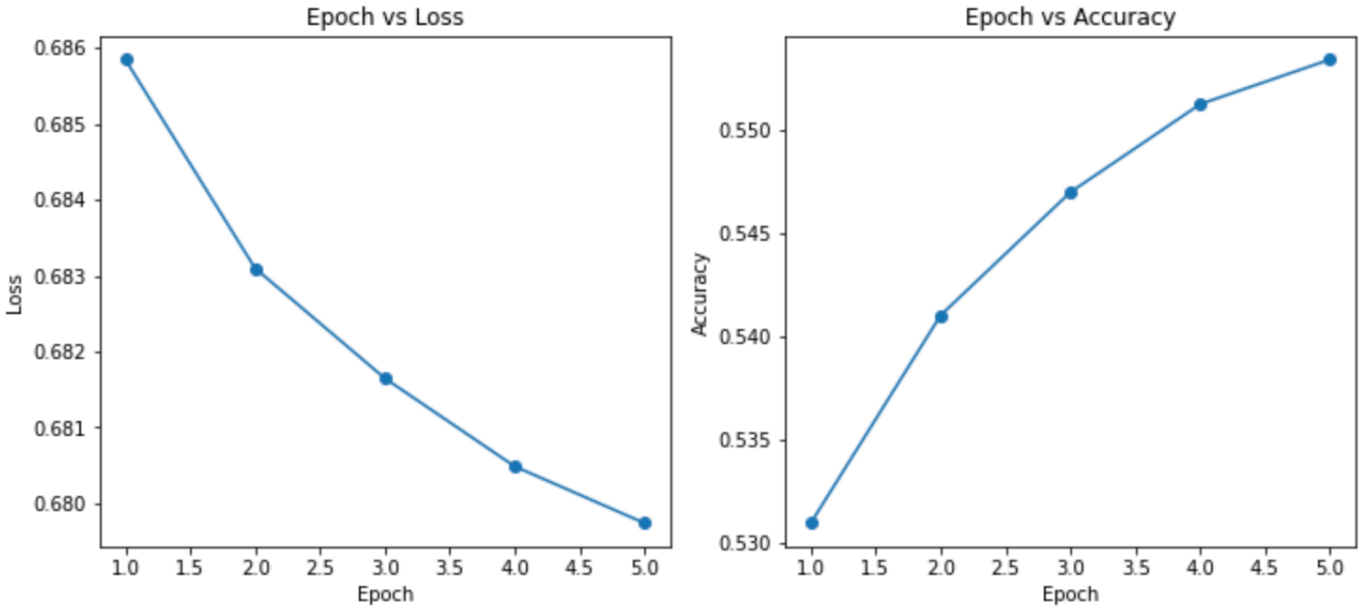
(2) Performance Analysis

With the selected hyperparameters, the RNN model achieved an accuracy of 54.77%, precision of 54.11%, and recall rate of 38.44% on the 2013 data. The accuracy and precision results are slightly higher than random guessing, but not significantly; the recall rate is very poor, indicating many positive samples were missed. This is also reflected in the confusion matrix.

The heatmap of the confusion matrix for the final classification is visualized as follows:



The training loss and accuracy curves are as follows:



From the confusion matrix, it can be seen that, consistent with the recall rate analysis, the model has a better prediction ability for negative samples, with an accuracy of about 70%; however, the prediction ability for positive samples is very poor, with an accuracy of about 35%. If the model's predictions for positive samples are inverted, the model's performance can be significantly improved.

From the training loss and accuracy curves, it can be seen that the training progresses steadily with iterations. The overall iteration count is low because we found that increasing the iterations caused the model's performance to decline, likely due to overfitting as the model is too large and lacks effective measures to prevent overfitting.

(3) Hyperparameter Adjustment Analysis

Adjustment	learning_rate	hidden_size	num_layers	num_epochs	dropout_rate	weight_decay	loss	accuracy
Initial	0.00005	80	3	5	0.5	0.01	0.6806	0.5466
learning_rate	0.0001	80	3	5	0.5	0.01	0.6798	0.5426
learning_rate	0.00001	80	3	5	0.5	0.01	0.6844	0.5295
num_layers	0.00005	80	4	5	0.5	0.01	0.6814	0.5414
dropout_rate	0.00005	80	3	5	0.2	0.01	0.6776	0.5474
dropout_rate	0.00005	80	3	5	0.7	0.01	0.6850	0.5099
dropout_rate	0.00005	80	3	5	0.1	0.01	0.6772	0.5485
dropout_rate	0.00005	80	3	5	0	0.01	0.6857	0.5372
weight_decay	0.00005	80	3	5	0	0.1	0.6843	0.5329
epoch	0.00005	80	3	10	0.1	0.01	0.6774	0.5458
weight_decay	0.00005	80	3	10	0.1	0.1	0.6775	0.5471
hidden_size	0.00005	200	3	10	0.1	0.1	0.6836	0.5344
hidden_size	0.00005	40	3	5	0.1	0.01	0.6780	0.5475
hidden_size	0.00005	200	3	5	0.1	0.01	0.6798	0.5418
hidden_size	0.00005	80	3	5	0.1	0.01	0.6773	0.5477

3. LSTM

(1) Model Structure Introduction

The LSTM model is a variant of the RNN model, developed to address the issues of gradient vanishing or explosion when dealing with long sequences. Compared to the GRU network, the LSTM model has a more complex structure and more input parameters, providing stronger memory capabilities for long sequences.

The final structure of the LSTM model we used is as follows:

```
EnhancedLSTMClassifier(
  (lstm): LSTM(6, 80, num_layers=5, batch_first=True, dropout=0.5, bidirectional=True)
  (fc_layers): Sequential(
    (0): Linear(in_features=160, out_features=80, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=80, out_features=2, bias=True)
  )
  (layer_norm): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
)
```

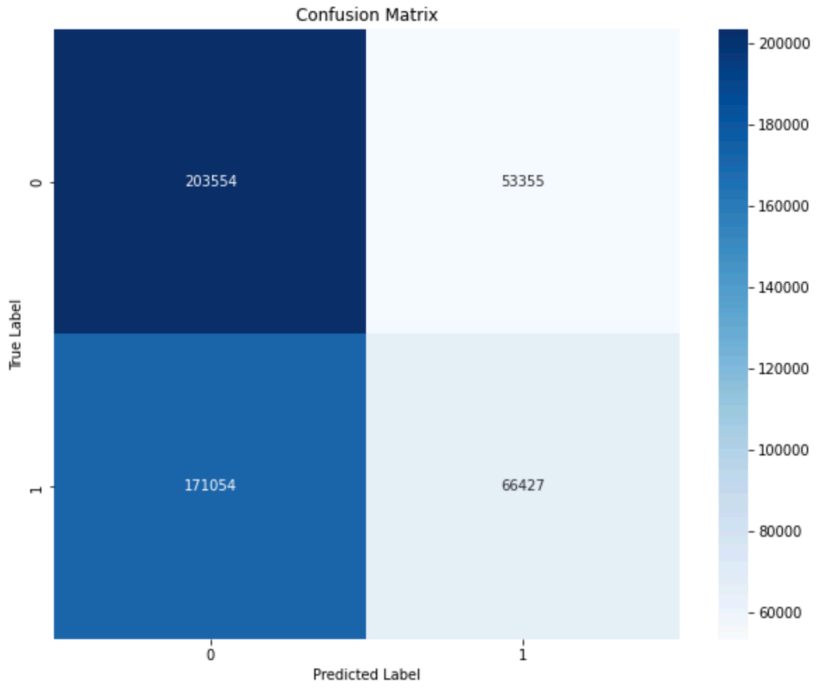
The final hyperparameters we used are as follows:

```
# Define model hyperparameters
hidden_size = 80
num_layers = 5
num_epochs = 5
dropout_rate = 0.5  # Set dropout rate
learning_rate = 0.0005  # Set learning rate
bidirection = True
weight_decay = 0.01
```

(2) Performance Analysis

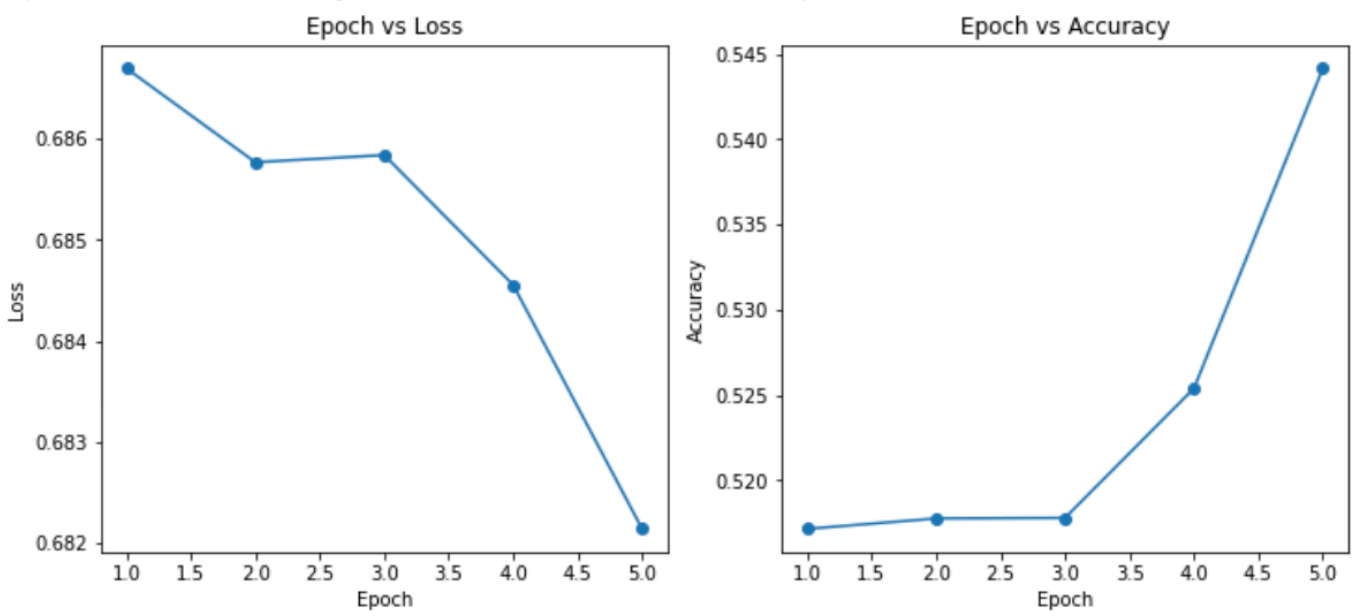
With the selected hyperparameters, the LSTM model achieved an accuracy of 54.61%, precision of 55.45%, and recall rate of 27.97% on the 2013 data. The results are similar to those of the RNN model, with a significantly lower recall rate. This suggests that an error in our processing resulted in very poor recognition of positive samples, which is amplified as the model complexity increases.

The heatmap of the confusion matrix for the final classification is visualized as follows:



It can be seen that the problem is similar to that in the RNN model, with good recognition of negative samples but very poor recognition of positive samples, classifying too many results as negative samples.

The training loss and accuracy curves are as follows:



It can be seen that performance improves significantly with increased training.

During training, many parameter combinations allowed for an accuracy of over 60% with 10 iterations, but all had severe overfitting issues on the test set, with the highest being close to 53%. This suggests that we need to modify the regularization method to avoid overfitting, but we could not find a more effective method online.

(3) Hyperparameter Adjustment Analysis

Adjustment	learning_rate	num_layers	hidden_size	num_epochs	weight_decay	dropout_rate	bidirection	loss	accuracy
Initial	0.00005	3	80	5	0.01	0.5	True	0.6774	0.5444
learning_rate	0.0005	3	80	5	0.01	0.5	True	0.6774	0.5440
learning_rate	0.005	3	80	5	0.01	0.5	True	0.6771	0.5484
num_layers	0.00005	5	80	5	0.01	0.5	True	0.6784	0.5433
learning_rate	0.0005	5	80	5	0.01	0.5	True	0.6772	0.5461
num_layers	0.0005	3	80	5	0.01	0.5	True	0.6781	0.5450
learning_rate	0.0001	5	80	5	0.01	0.5	True	0.6777	0.5436
num_epoch	0.0001	5	80	15	0.01	0.5	True	0.6797	0.5455
num_epoch	0.0005	5	80	15	0.01	0.5	True	0.6917	0.5367
weight_decay	0.0005	5	80	15	0.04	0.5	True	0.6801	0.5416
num_epoch	0.0005	5	80	10	0.01	0.5	True	0.6821	0.5439
learning_rate	0.001	5	80	5	0.01	0.5	True	0.6924	0.5196

4. GRU

(1) Model Structure Introduction

The GRU model is a variant of the RNN model, developed to address the issues of gradient vanishing or explosion when dealing with long sequences. Compared to the LSTM network, the GRU model has faster computation speed and better capture of short-term dependencies.

The final structure of the GRU model we used is as follows:

```
EnhancedGRUClassifier(
  (gru): GRU(6, 80, num_layers=3, batch_first=True, dropout=0.3, bidirectional=True)
  (fc_layers): Sequential(
    (0): Linear(in_features=160, out_features=80, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=80, out_features=2, bias=True)
  )
  (layer_norm): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
)
```

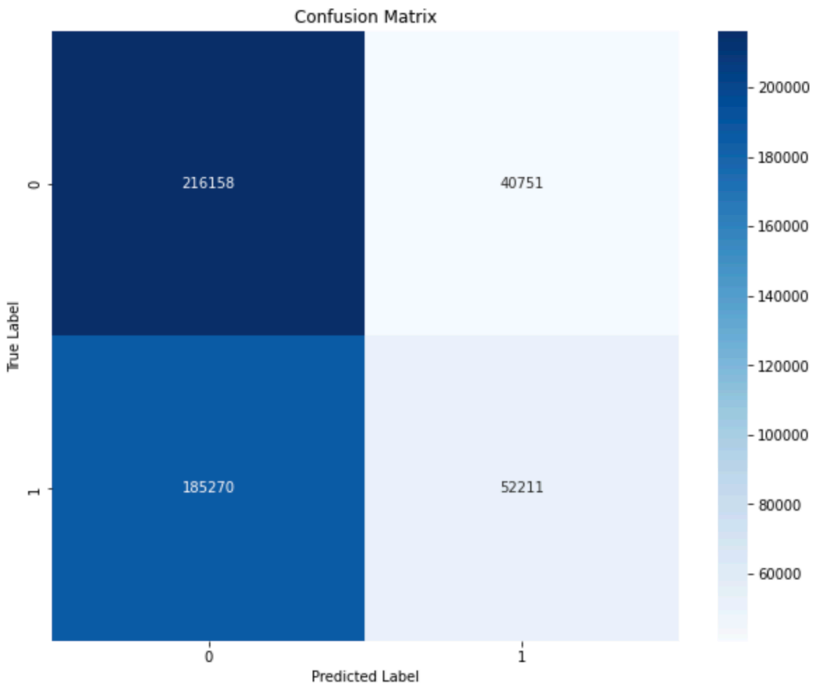
The final hyperparameters we used are as follows:

```
# Define model hyperparameters
hidden_size = 80
num_layers = 3
num_epochs = 10
dropout_rate = 0.3  # Set dropout rate
learning_rate = 0.0001  # Set learning rate
bidirection = True
weight_decay = 0.01
```

(2) Performance Analysis

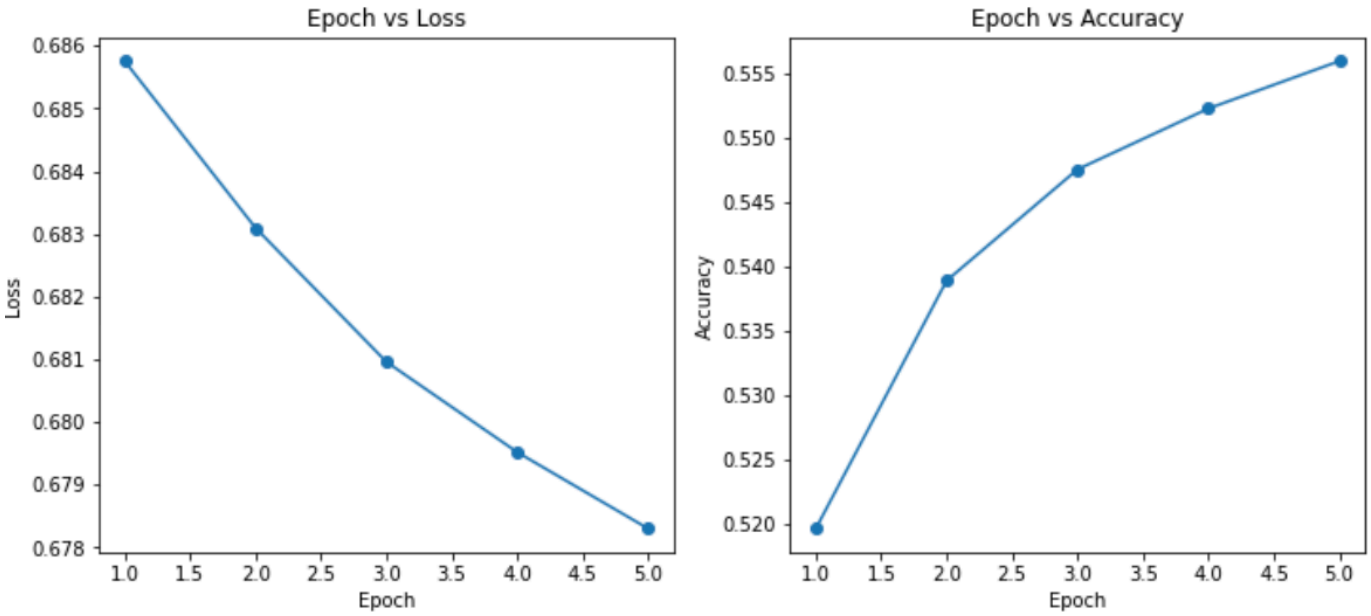
With the selected hyperparameters, the GRU model achieved an accuracy of 54.66%, precision of 56.16%, and recall rate of 21.98% on the 2013 data. The accuracy and precision results are higher than random guessing but not significantly; the recall rate is very poor, indicating many positive samples were missed. This is also reflected in the confusion matrix.

The heatmap of the confusion matrix for the final classification is visualized as follows:



It can be seen that the overall situation is similar to that of the RNN network. From the observation of the confusion matrix, we can draw the same conclusion, i.e., the model performs well on negative samples with an accuracy of about 85%, consistent with the recall rate analysis. However, for positive samples, the model's performance is very poor, with an accuracy of only 20%. If we reverse the model's predictions for positive samples, it can significantly improve performance.

The training loss and accuracy curves are as follows, showing a steady improvement with training:



The reason for not using more iterations here is the same as before: we cannot address the overfitting issue that arises with more iterations.

(3) Hyperparameter Adjustment Analysis

Adjustment	learning_rate	num_layers	hidden_size	num_epochs	weight_decay	dropout_rate	bidirection	loss	accuracy
Initial	0.00005	2	80	5	0.01	0.5	True	0.6778	0.5441
Learning_rate	0.0005	2	80	5	0.01	0.5	True	0.6788	0.5389
Learning_rate	0.000005	2	80	5	0.01	0.5	True	0.6796	0.5330
num_layers	0.00005	5	80	5	0.01	0.5	True	0.6797	0.5301
learning_rate	0.0002	5	80	5	0.01	0.5	True	0.6839	0.5395
num_layers	0.0002	3	80	5	0.01	0.5	True	0.6784	0.5428
dropout_rate	0.0002	3	80	5	0.01	0.3	True	0.6788	0.5448
num_epochs	0.0001	3	80	15	0.01	0.3	True	0.6833	0.5437
dropout_rate	0.0001	3	80	15	0.01	0.5	True	0.6788	0.5448
num_epochs	0.00005	2	80	15	0.01	0.5	True	0.6772	0.5485
num_layers	0.0001	3	80	15	0.01	0.5	True	0.6794	0.5426
weight_decay	0.0001	3	80	15	0.04	0.5	True	0.6788	0.5443
num_epoch	0.0002	3	80	10	0.01	0.3	True	0.6832	0.5438
learning_rate	0.0001	3	80	10	0.01	0.3	True	0.6829	0.5425

5. Comparative Analysis of the Three Models

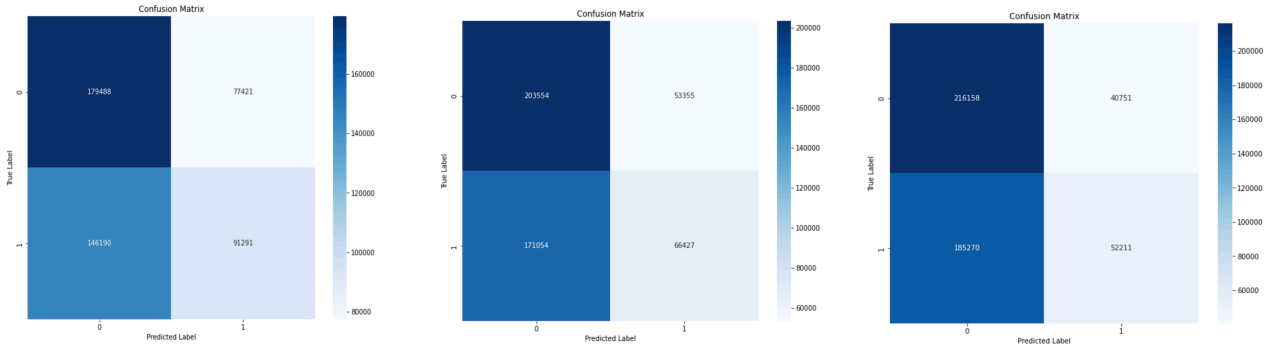
In this section, we compare and analyze the different results of the three models, trying to explain the differences and commonalities.

The quantitative results of the three models for actual classification are as follows:

	Accuracy	Precision	Recall Rate
RNN	54.77%	54.11%	38.44%
LSTM	54.61%	55.45%	27.97%
GRU	54.66%	56.16%	21.98%

From the table, we can see that although the LSTM and GRU models are improvements over the RNN model, the optimal performance achieved in our parameter tuning results does not show significant differences. The precision and recall rates also show similar trends; there is no noticeable difference in precision, and the recall rate for LSTM and GRU is significantly lower than RNN. It is hypothesized that the recall rate is inversely proportional to the complexity of the model. This may be due to the overall data size and model being large, and our lack of effective means to suppress overfitting.

The comparison of the confusion matrix heatmaps for the final prediction results of the three models is as follows, with the RNN results on the left, the LSTM results in the middle, and the GRU results on the right:



From the heatmaps, we can see that all three models share a common issue: they have good recognition ability for negative samples but very poor recognition ability for positive samples, overall classifying too many samples as negative.

Comparative Analysis

In this section, we comprehensively compare the different models based on different types of data in Task 1 and Task 2, analyzing their advantages and disadvantages.

Before the actual operation, our expectation for the two tasks was that Task 2 should significantly outperform Task 1 in terms of performance. This is because the difference in the complexity of the models between the two tasks is not substantial, and the image information used in Task 1 is a "dimensionality reduction" of the data in Task 2, with considerable information loss. Since they need to predict the same result, Task 2 should have more abundant data and thus achieve better prediction results.

In practice, however, the effect of RNN was not as good as that of the CNN network. This may be due to our poor design of the RNN, but it may also be related to the characteristics of the Chinese A-share market. The large volatility of the Chinese A-share market reduces the correlation of stock price and volume data sequences, which affects the RNN's ability to effectively capture the dependency features between long sequence elements. In contrast, the CNN, which can capture local features of the images, performs better.

1. CNN & Images

Advantages:

1. **Suitable for Image Data Processing:** CNN performs exceptionally well in handling image data. It can effectively extract features from images, including edges, shapes, and textures, helping to recognize complex visual patterns.
2. **Feature Recognition:** Capable of effectively capturing spatial features and local patterns in image data, making it suitable for processing stock candlestick charts and other form data.
3. **Parameter Sharing and Sparse Connections:** CNN's parameter sharing mechanism and sparse connections significantly reduce the number of model parameters, lowering the risk of overfitting and improving efficiency.

Disadvantages:

1. **Prone to Overfitting:** If the training data is insufficient or the model complexity is too high, CNN is prone to overfitting, leading to poor generalization on new data.
2. **Black-box Model:** Deep CNN models are usually considered black-box models, making it difficult to interpret the decision-making process and feature extraction methods within the model, limiting its interpretability.
3. **Data Preprocessing:** May require more preprocessing for some unstructured data.
4. **Inadequate for Long-term Sequence Processing:** May not perform as well as RNN for data with strong long-term dependencies.

2. RNN & Time Series Data

Advantages:

1. **Handling Sequential Data:** RNNs are inherently suited for processing sequential data as they retain a memory of previous time steps' information while processing each new input, making them effective for time series data.
2. **Flexible Input and Output Lengths:** RNNs can handle variable-length sequences, making them very effective in tasks like natural language processing and time series prediction.
3. **Context Understanding:** RNNs can capture long-term dependencies in sequential data, thus performing well in understanding context and recognizing sequential patterns.
4. **Effective Memory:** Capable of effectively remembering and utilizing historical data, suitable for processing time-series data like stock prices and volumes.

Disadvantages:

1. **Local Processing:** May not perform as well as CNN for form data, especially for data that requires capturing local patterns.
2. **Gradient Vanishing/Exploding:** RNNs are prone to gradient vanishing or exploding issues during training, especially for long-term dependencies, which can lead to difficult or unstable model training.
3. **Low Computational Efficiency:** RNNs process long sequences sequentially, leading to low computational efficiency.

In summary, while CNNs are superior in handling image data and extracting spatial features, RNNs excel in processing sequential data and capturing long-term dependencies. The choice between them should be based on the nature of the data and the specific requirements of the task at hand.