Project1

田田 经12计18 2021011048

彭扬达 经12计18 2021011054

何牧桦 未央水木11 2021012553

一、Task1

（一）数据划分

将加载的数据集划分为训练集、验证集（80%-20%分割）和测试集，并使用 `torch.utils.data.DataLoader` 将数据转换为批量数据加载器（`train_loader`、`val_loader` 和 `test_loader`）。

（二）模型结构

定义了一个 CNN 模型，其中包括两个卷积层（`self.conv1` 和 `self.conv2`）、最大池化层（`self.pool`）、两个全连接层（`self.fc1` 和 `self.fc2`）以及 Sigmoid 激活函数。

使用 Adam 优化器或 SGD 优化器更新模型的参数，使用二元交叉熵损失 BCELoss 来衡量二分类任务中模型输出与真实标签之间的差异。

最终我们的模型结构为：

```
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()
)
```

（三）超参数与模型结构调整

调整对象	batch	epoch	learning_rate	accuracy
初始值	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

在迭代的时候我们发现，较小的 lr 更容易按照正确的方向去优化，不易陷入“陷阱”当中。

从迭代曲线中我们也可以看出，epoch 在10以内时，随着迭代次数的增加，曲线是逐渐收敛的。

使用batch = 64，epoch = 10，lr = 0.001这组参数继续进行模型结构的调整：

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

卷积层角度，卷积核大小决定了每个卷积核在输入图像上感受野的大小，较小的卷积核通常用于捕获局部特征，而较大的卷积核可以捕获更大的特征结构，更多的卷积核可以捕获更多不同类型的特征。

池化层角度，最大池化保留最显著的特征，平均池化对特征进行平均处理。池化核大小决定了池化操作的窗口大小，控制了输出特征图的尺寸。较大的池化核会减小特征图的空间维度，从而降低模型复杂度。

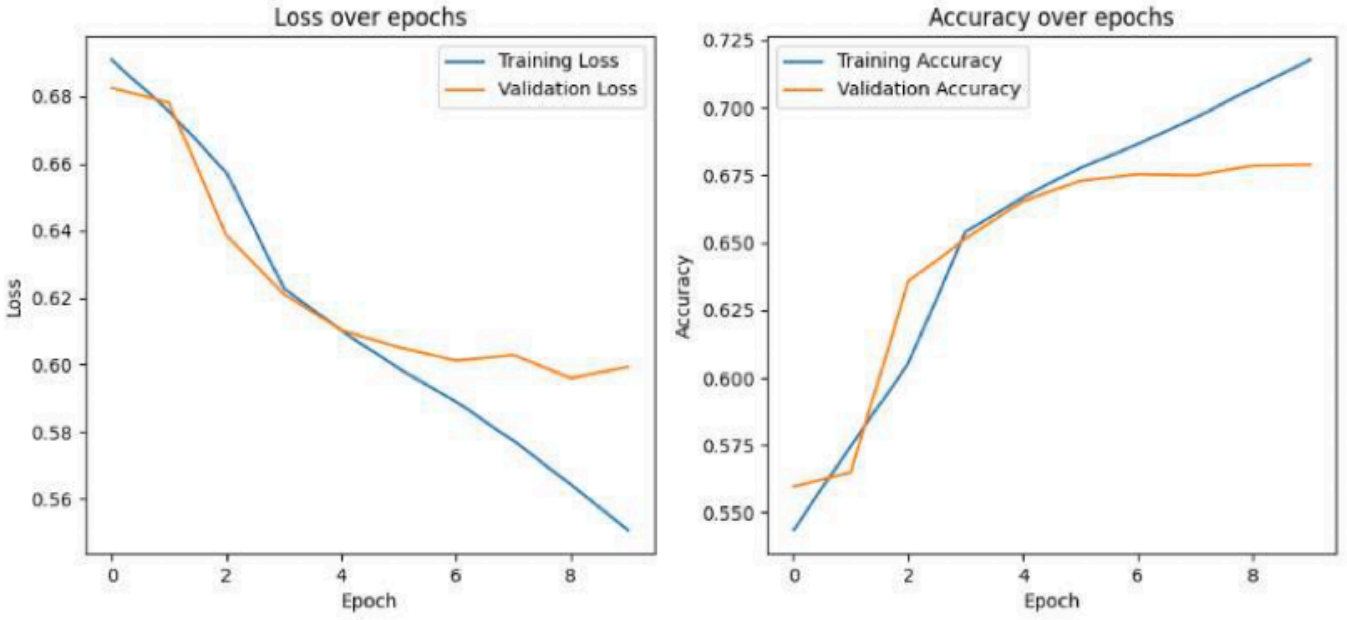
全连接层角度，隐藏单元数控制了每个全连接层的输出维度，影响模型的表达能力。更多的隐藏单元可以捕获更复杂的特征关系，但也可能导致过拟合。

使用 conv1,2 = 3,32,3—32,64,3; pool = 2,2; fc1,2 = 64x45x24, 128—128,1这组参数继续调整：

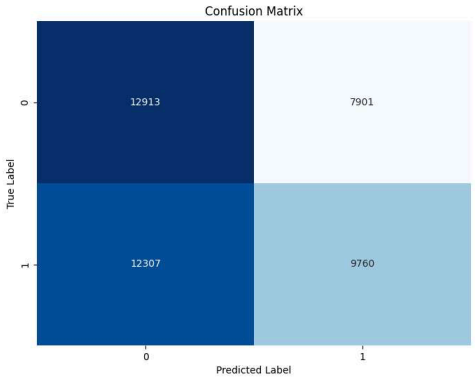
optimizer	loss	Accuracy
Adam	0.5993	0.5569
SGD	0.6218	0.5361

（四）结果分析

画出使用 Adam 优化器、BCELoss 损失函数、batch = 64、epoch = 10、lr = 0.001、conv1,2 = 3,32,3—32,64,3、pool = 2,2、fc1,2 = 64x45x24, 128—128, 1的 loss 和 accuracy曲线如下图所示：



```
Epoch 1:
Training Loss: 0.6910, Training Accuracy: 0.5435
Validation Loss: 0.6827, Validation Accuracy: 0.5597
Epoch 2:
Training Loss: 0.6756, Training Accuracy: 0.5747
Validation Loss: 0.6782, Validation Accuracy: 0.5650
Epoch 3:
Training Loss: 0.6574, Training Accuracy: 0.6052
Validation Loss: 0.6386, Validation Accuracy: 0.6356
Epoch 4:
Training Loss: 0.6229, Training Accuracy: 0.6540
Validation Loss: 0.6211, Validation Accuracy: 0.6515
Epoch 5:
Training Loss: 0.6103, Training Accuracy: 0.6668
Validation Loss: 0.6104, Validation Accuracy: 0.6654
Epoch 6:
Training Loss: 0.5989, Training Accuracy: 0.6778
Validation Loss: 0.6053, Validation Accuracy: 0.6729
Epoch 7:
Training Loss: 0.5890, Training Accuracy: 0.6864
Validation Loss: 0.6012, Validation Accuracy: 0.6754
Epoch 8:
Training Loss: 0.5774, Training Accuracy: 0.6963
Validation Loss: 0.6029, Validation Accuracy: 0.6749
Epoch 9:
Training Loss: 0.5643, Training Accuracy: 0.7071
Validation Loss: 0.5960, Validation Accuracy: 0.6786
Epoch 10:
Training Loss: 0.5506, Training Accuracy: 0.7177
Validation Loss: 0.5993, Validation Accuracy: 0.6790
测试集准确率: 0.5569
```



通过观察混淆矩阵我们发现，该矩阵有较高的对角线值，表明我们的 CNN 模型在该类别上表现良好。

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

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

二、Task2

我们分别尝试了 LSTM，GRU 和普通 RNN 这三种模型和各种超参数。在这一部分我们记录我们尝试的过程，分享我们的一些思考和反思。

（一）数据划分

我们按照项目说明中的要求，使用2010年到2012年这三年的数据作为训练集，使用2013年的数据作为测试集，不使用验证集。使用每个交易日过去 22 天的数据，预测未来五天的价格变化方向。

考虑到本次的数据是时序数据，并且我们是采用过去一段时间的数据预测未来五天的变化，因此如果我们使用交叉验证的方式的话，很容易会将未来数据投入到模型的训练中，从而导致非常严重的过拟合。因此，我们直接按照要求对数据进行划分预处理。

我们原本采用了 MInMaxScalar 归一化函数，针对课程提供的经过数据特征预处理之后的数据再次进行归一化操作，但发现随着训练模型的预测结果全部变为0。只进行一次预处理则正常训练，推测是因为数据如若已经经过了归一化处理，反复处理的话可能会使得不同特征之间的大小关系被进一步抹去，导致模型在学习过程中失去了原始数据的特征。两次处理后数据的范围变得更小，导致所有数据都趋近于0，丢失原始数据的信息，使得模型无法有效地学习。此外很可能导致了梯度消失问题，因为数据被压缩到一个非常小的范围内，导致更新时梯度变得非常小，从而几乎无法对模型参数进行调整，导致模型恒输出零。

（二）普通RNN

1、模型结构介绍

RNN 模型是为了处理序列数据而被发明出来的模型，通过它的记忆功能来有效处理时间序列数据。

我们采用的 RNN 模型的最终结构如下：

```
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)
)
```

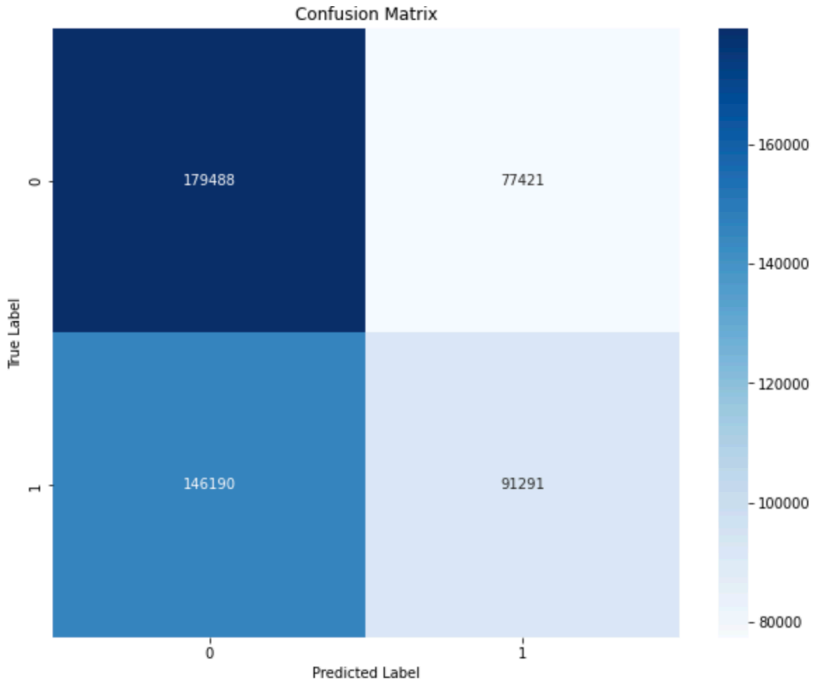
最终我们采用的超参数如下：

```
# 定义模型超参数
hidden_size = 80
num_layers = 3
num_epochs = 5
dropout_rate = 0.1  # 设置dropout率
learning_rate = 0.00005  # 设置学习率
weight_decay = 0.01
```

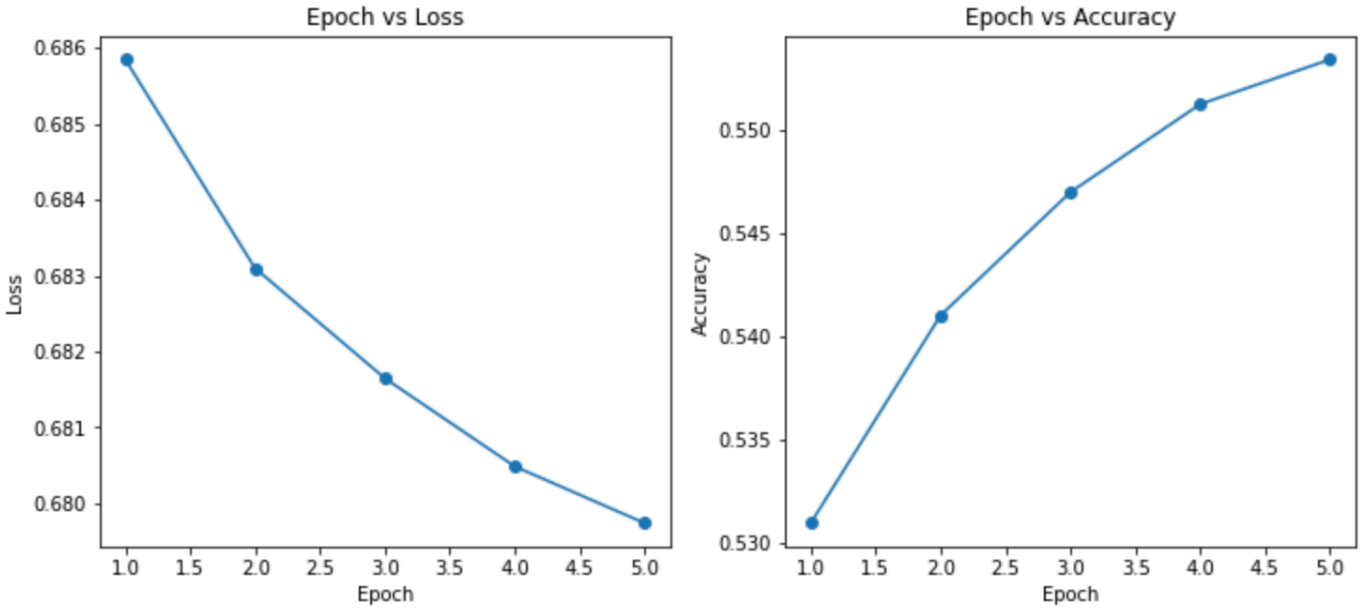
2、性能分析

在最终选取的超参数下，RNN 模型在2013年的数据上的预测 accuracy 为54.77%，precision 为54.11%，recall rate 为38.44%。准确率和精确率的结果高于随机猜测，但是并不是非常明显；而召回率的结果非常糟糕，说明有很多的正样本被错过了，最终的混淆矩阵中的结果也说明了这一点。

模型最终分类情况的混淆矩阵热力图可视化结果如下：



训练中 loss 和 accuracy 的变化曲线如下：



从混淆矩阵中可以看出，和召回率的分析结果一样，模型对于负样本的情况有比较好的预测能力，正确率大约在70%左右；但模型对于正样本的预测能力非常糟糕，正确率为35%左右，如果将模型对于正样本的预测结果取反，将可以明显提升模型的性能。

从训练中 loss 和 accuracy 的变化曲线可以看出，在训练中还是随着迭代稳步提升的。整体的迭代次数较低，是因为我们发现增加迭代次数后，模型的性能反而开始下降，可能是因为模型太大，我们缺乏有效的抑制过拟合的手段，从而随着迭代次数的增加出现了明显的过拟合。

3、超参数调整分析

调整对象	learning_rate	hidden_size	num_layers	num_epochs	dropout_rate	weight_decay	loss	accuracy
	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

（三）LSTM

1、模型结构介绍

LSTM 模型是 RNN 模型的变体，为了解决处理长序列时容易出现的梯度消失或爆炸的问题而研发。相较于 GRU 网络而言，LSTM 模型有着更加复杂的结构和更多的输入参数，针对长序列有着更强的记忆能力。

我们采用的 LSTM 模型的最终结构如下：

```
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)
)
```

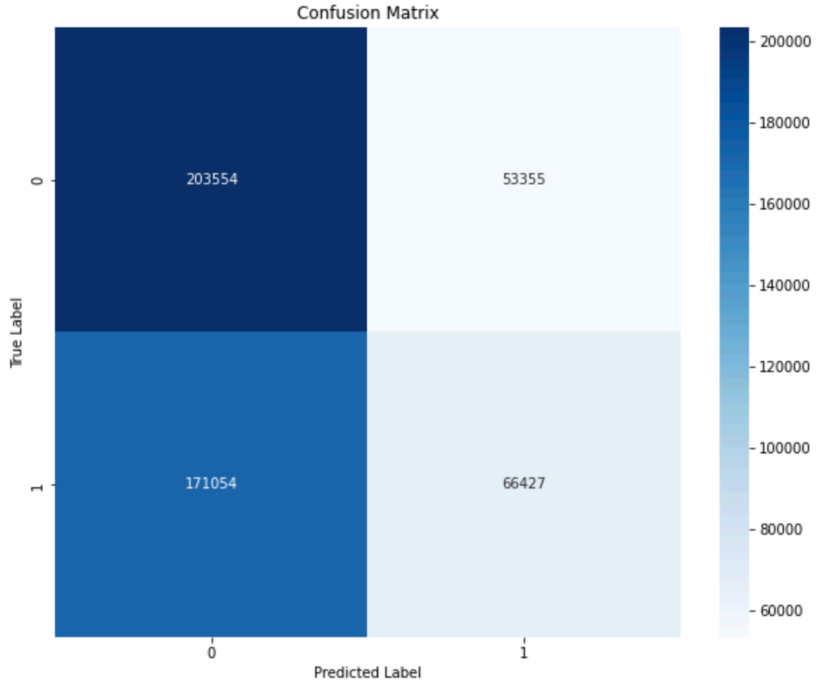
最终我们采用的超参数如下：

```
# 定义模型超参数
hidden_size = 80
num_layers = 5
num_epochs = 5
dropout_rate = 0.5  # 设置dropout率
learning_rate = 0.0005  # 设置学习率
bidirection = True
weight_decay = 0.01
```

2、性能分析

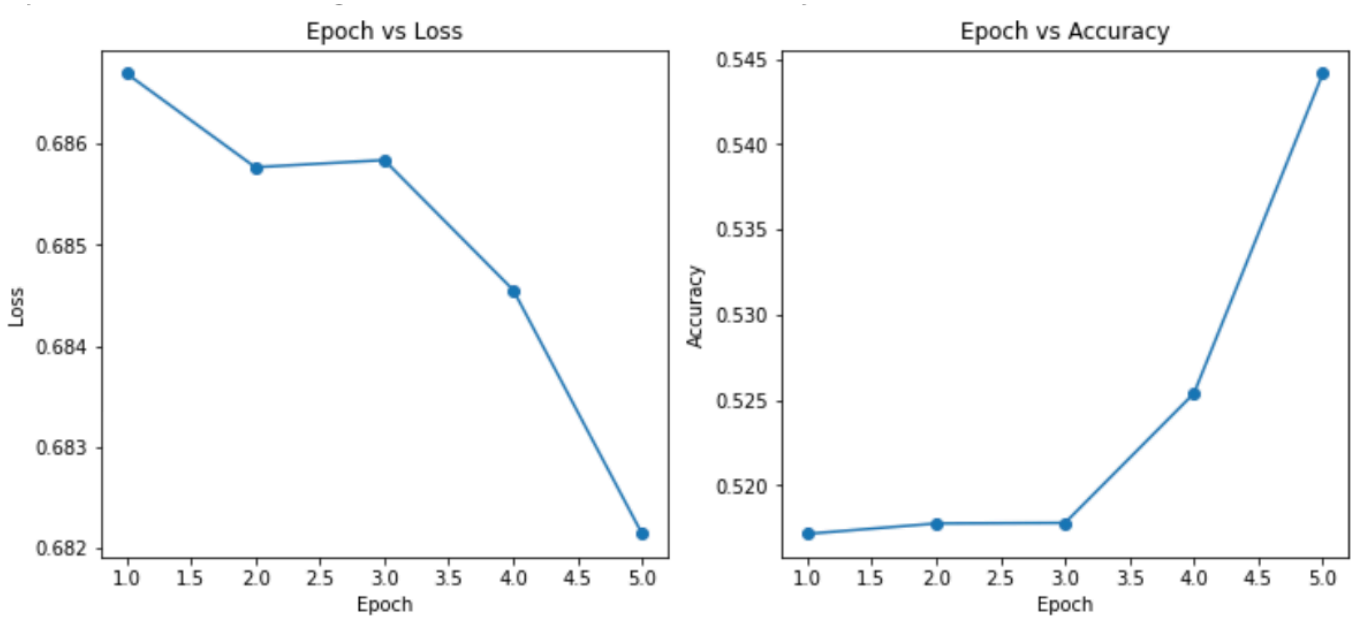
在最终选取的超参数下，LSTM 模型在2013年的数据上的预测 arruracy 为54.61%，precision 为55.45%，recall rate 为27.97%。结果整体和 RNN 的情况类似，只是有着显著更低的召回率。推测是因为我们的某个错误处理使得对于正样本 的识别非常糟糕，随着模型复杂度的增加这个错误会被进一步放大。

模型最终分类情况的混淆矩阵热力图可视化结果如下：



可以看出，问题和 RNN 中的类似，有着对负样本良好的识别和对正样本非常糟糕的识别，将太多的结果识别成了负样本。

训练中 loss 和 accuracy 的变化曲线如下：



可以看出，随着训练次数的提升，性能有着比较明显的提升。

在训练过程中，其实出现了很多参数组合可以让迭代10次时就出现高于60%的 accuracy，但他们在测试集上的表现都存在非常严重的过拟合，最高的也就是接近53%。推测在这里我们需要修改正则化的方式来避免过拟合，但我们没能在网络上查找到更加有效的方式。

3、超参数调整分析

调整对象	learning_rate	num_layers	hidden_size	num_epochs	weight_decay	dropout_rate	bidirection	loss	accuracy
初始	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

（四）GRU

1、模型结构介绍

GRU 模型是 RNN 模型的变体，为了解决处理长序列时容易出现的梯度消失或爆炸的问题而研发。相较于 LTSM 网络而言，GRU 模型有着更加快速的运算速度和对于短期依赖关系更好的捕捉。

我们采用的 GRU 模型的最终结构如下：

```
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)
)
```

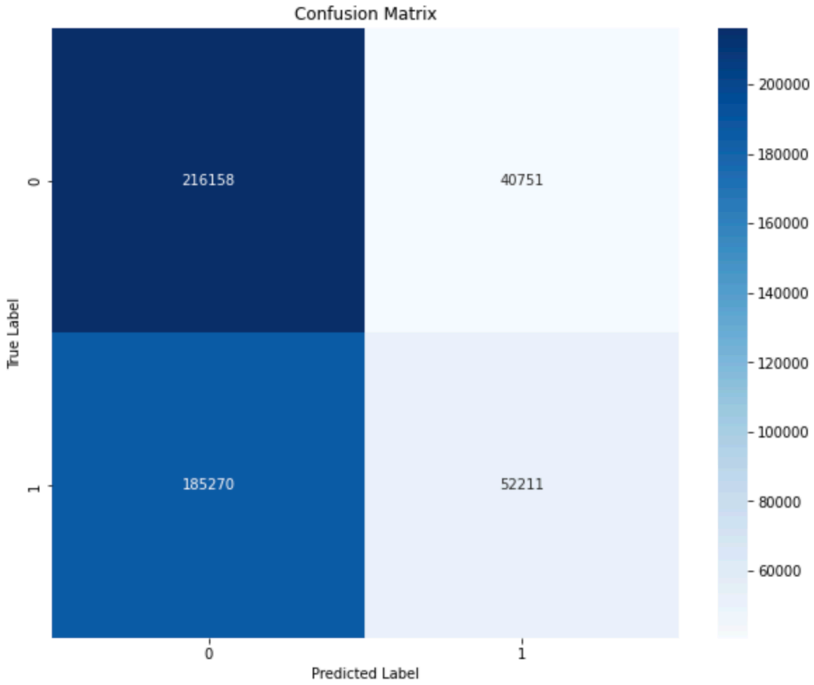
最终我们采用的超参数如下：

```
# 定义模型超参数
hidden_size = 80
num_layers = 3
num_epochs = 10
dropout_rate = 0.3  # 设置dropout率
learning_rate = 0.0001  # 设置学习率
bidirection = True
weight_decay = 0.01
```

2、性能分析

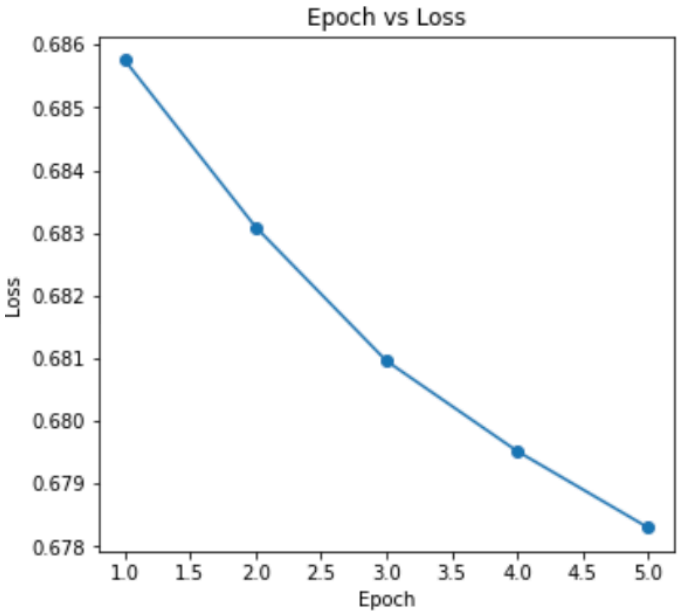
在最终选取的超参数下，GRU 模型在2013年的数据上的预测 accuracy 为54.66%，precision 为56.16%，recall rate 为21.98%。准确率和精确率的结果高于随机猜测，但是并不是非常明显；而召回率的结果非常糟糕，说明有很多的正样本被错过了，最终的混淆矩阵中的结果也说明了这一点。

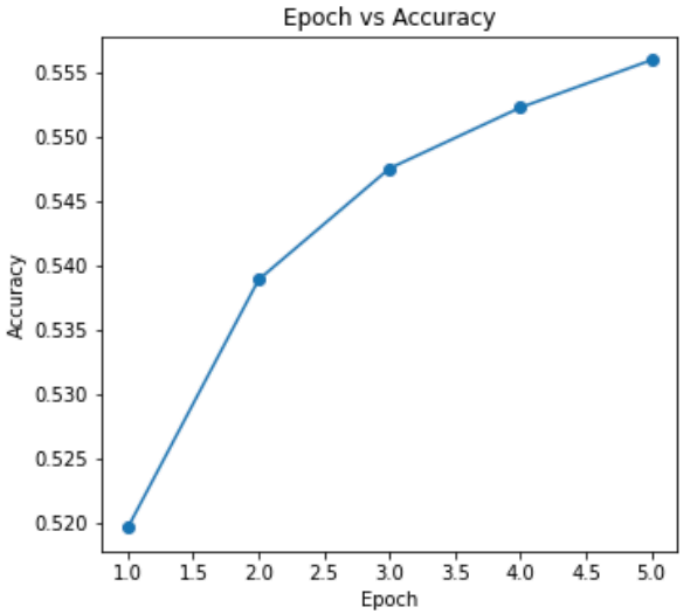
模型最终分类情况的混淆矩阵热力图可视化结果如下：



可以看出，整体情况和 RNN 网络的类似。从混淆矩阵的观察可以得出相同的结论，即模型在负样本方面表现良好，正确率约为85%，与召回率分析的结果相符。然而，在正样本方面，模型的表现非常糟糕，仅有20%的正确率。如果我们对模型在正样本上的预测结果进行反转，可以显著提高其性能。

在训练过程中上 loss 和 accuracy 的变化过程如下，可以看出同样是随着训练稳定提升。





在这里没有采用更多的迭代次数的原因和先前相同：我们无法处理迭代次数增加后带来的过拟合问题。

3、超参数调整分析

调整对象	learning_rate	num_layers	hidden_size	num_epochs	weight_decay	dropout_rate	bidirection	loss	accuracy
初始	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

（五）三种模型比较分析

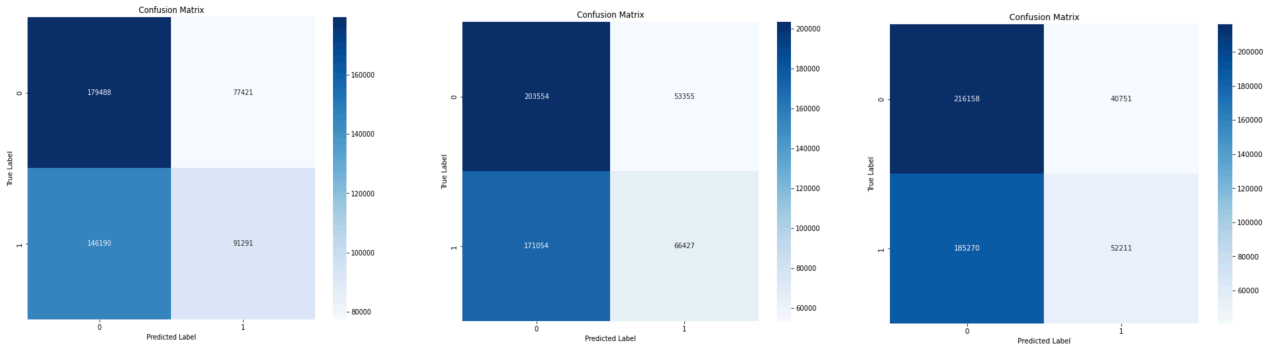
在这一部分，我们比较分析三个模型不同的结果，尝试解释差别和共性。

三种模型实际分类的定量结果对比如下：

	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%

从中我们可以发现，尽管 LSTM 和 GRU 模型是针对 RNN 模型的提升，但是在我们调参的结果中，取得的最优性能并没有比较显著的差距。精确率和召回率的情况也类似，精确率没有明显差异，召回率 LSTM 和 GRU 相比 RNN 反而有非常明显的下降，推测可能是召回率和模型的复杂程度成反比。推测是因为整体数据量和模型比较大，而我们缺乏足够有效的抑制过拟合的问题所致。

三种模型最终预测结果的混淆矩阵热力图对比如下，左图中是 RNN 的结果，中间是 LSTM 的结果，右边是 GRU 的结果：



从中我们可以发现，三种模型存在非常共性的问题：即有着比较优秀的对于负例的识别能力，但是对于正例的整体识别能力非常糟糕，整体而言将过多的样本识别为了负例。

三、比较分析

在这一部分，我们综合比较 Task1 和 Task2 中两个基于不同类型数据的不同模型，分析他们的优缺点。

在开始实际操作之前，我们对于两个任务的预期是：Task2 应该在性能上明显优于 Task1 结果，因为两者的模型的复杂程度粗略来讲差异不大，而 Task1 中使用的图像信息是由 Task2 中的数据“降维”得到的，有着比较多的信息损失，且他们需要预测相同的结果。因此，我们认为 Task2 有着比 Task1 更加丰富的数据，理应取得更加优秀的预测结果。

实际操作上，RNN 的效果反而不如 CNN 网络的结果。这可能是由于我们的 RNN 设计的不够好，但或许也和中国A股的特征有关：中国A股的波动很大，这导致实际上股票量价数据序列相关性降低，这就影响了 RNN 有效捕捉长序列元素之间相关依赖特征长处的发挥，相比之下，能够捕捉图形局部特征的 CNN 表现更为优秀也就不足为奇了。

（一）CNN & 图像

优点：

1. **适用于图像数据处理：** CNN 在处理图像数据方面表现优异。它能够有效地提取图像中的特征，包括边缘、形状、纹理等信息，有助于识别复杂的视觉模式。
2. **图片特征识别：** 能够有效地捕捉图像数据中的空间特征和局部模式，适合处理股票K线图这种形态数据。
3. **参数共享和稀疏连接：** CNN 的参数共享机制和稀疏连接方式有效地减少了模型的参数量，降低了过拟合的风险，并且使得模型更具有效率。

缺点：

1. **容易过拟合：** 如果训练数据不足或模型复杂度过高，CNN容易发生过拟合，导致模型在新数据上的泛化能力较差。
2. **黑盒模型：** 深层CNN模型通常被视为黑盒模型，难以解释模型内部的决策过程和特征提取方式，限制了模型的可解释性。

3. **数据预处理**：对于一些非结构化数据，可能需要更多的预处理工作。
4. **长期序列处理不足**：对于长期依赖性较强的序列数据，可能不如 RNN 效果好。

（二）RNN & 时序数据

优点：

1. **处理序列数据：** RNN天生适合处理序列数据，因为它们在处理每个时间步的输入时都会保留一种记忆，这种记忆可用于处理前面时间步的信息。
2. **灵活的输入和输出长度：** RNN可以处理不同长度的序列输入和输出，这使得它们在自然语言处理、时间序列预测等任务中非常有效。
3. **上下文理解：** RNN可以捕获序列数据中的长期依赖关系，因此在理解上下文、识别序列模式方面表现良好。
4. **有效记忆**：能够对历史数据进行有效的记忆和利用，适合处理量价数据等按时间顺序排列的数据。

缺点：

1. **局部处理**：在面对一些形态数据时可能表现不如 CNN，特别是对于需要捕捉局部模式的数据。
2. **梯度消失/爆炸：** RNN训练时容易出现梯度消失或梯度爆炸问题，尤其是对于长期依赖的序列，这可能导致难以训练或者不稳定的模型。
3. **计算效率低：** RNN在处理长序列时，每个时间步都需要依次计算，导致计算效率低下。