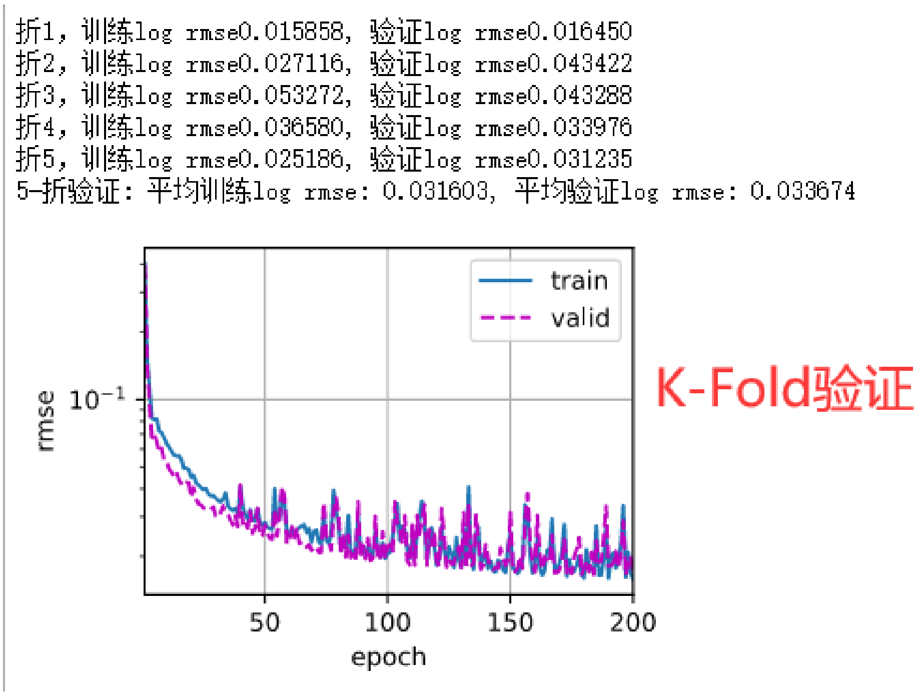
模型训练与结果分析

房价预测

线性预测模型

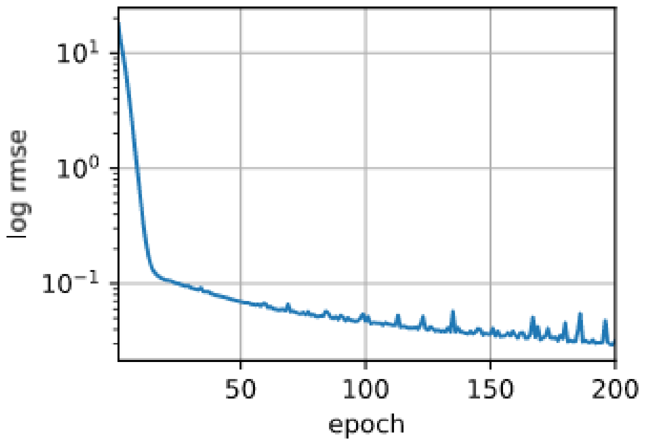
```python
loss = nn.MSELoss()
in_features = train_features.shape[1]

def get_net():
    net = nn.Sequential(nn.Linear(in_features, 1))
    # 模型参数初始化
    for param in net.parameters():
        nn.init.normal_(param, mean=0, std=0.01)
    return net
```



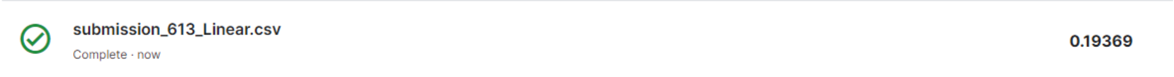
```
折1，训练log rmse0.015858, 验证log rmse0.016450
折2，训练log rmse0.027116, 验证log rmse0.043422
折3，训练log rmse0.053272, 验证log rmse0.043288
折4，训练log rmse0.036580, 验证log rmse0.033976
折5，训练log rmse0.025186, 验证log rmse0.031235
5-折验证: 平均训练log rmse: 0.031603, 平均验证log rmse: 0.033674
```

K-Fold验证

```
训练log rmse: 0.031656
```



训练的log RMSE

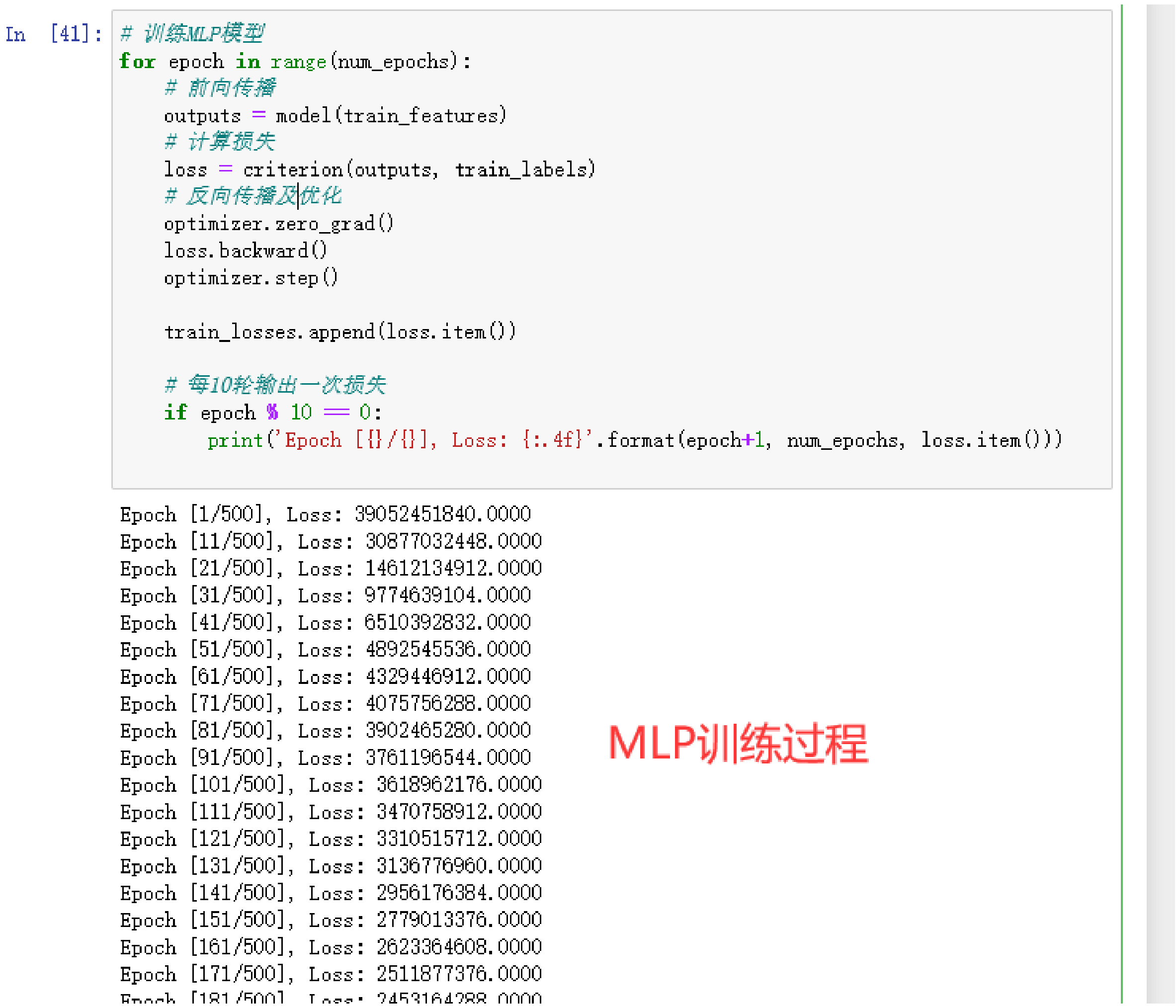


submission_613_Linear.csv
Complete · now

0.19369

普通MLP预测模型

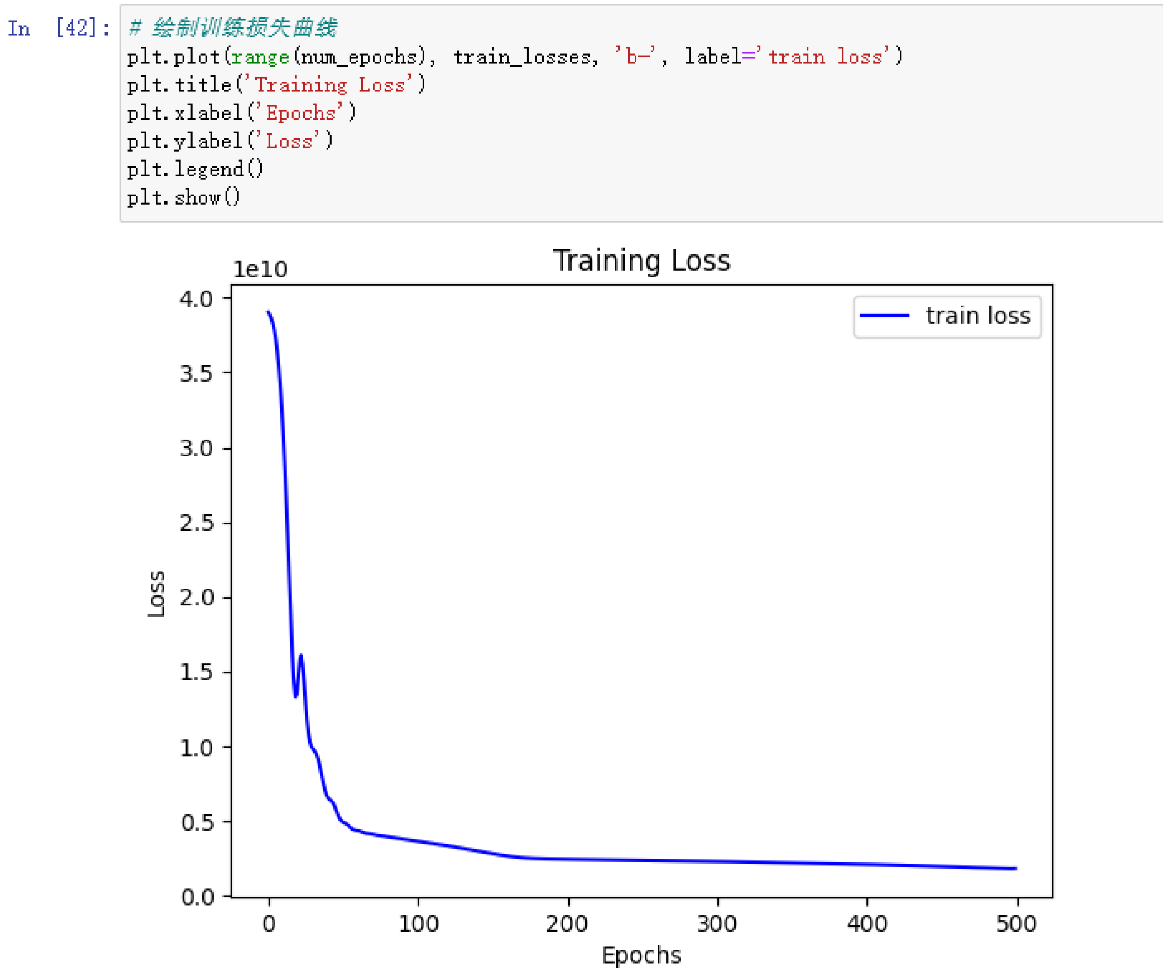
```
# 设置超参数
input_dim = train_features.shape[1]
output_dim = 1
hidden_dim = 512
lr = 0.001
num_epochs = 500

# 初始化模型、损失函数以及优化器
model = Net(input_dim, hidden_dim, output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

# 训练MLP模型
for epoch in range(num_epochs):
    # 前向传播
    outputs = model(train_features)
    # 计算损失
    loss = criterion(outputs, train_labels)
    # 反向传播及优化
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())

    # 每10轮输出一次损失
    if epoch % 10 == 0:
        print('Epoch [{}/{}], Loss:
{:.4f}'.format(epoch+1, num_epochs, loss.item()))
```



```
In  [41]: # 训练MLP模型
          for epoch in range(num_epochs):
              # 前向传播
              outputs = model(train_features)
              # 计算损失
              loss = criterion(outputs, train_labels)
              # 反向传播及优化
              optimizer.zero_grad()
              loss.backward()
              optimizer.step()

              train_losses.append(loss.item())

              # 每10轮输出一次损失
              if epoch % 10 == 0:
                  print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.item()))
```

```
Epoch [1/500], Loss: 39052451840.0000
Epoch [11/500], Loss: 30877032448.0000
Epoch [21/500], Loss: 14612134912.0000
Epoch [31/500], Loss: 9774639104.0000
Epoch [41/500], Loss: 6510392832.0000
Epoch [51/500], Loss: 4892545536.0000
Epoch [61/500], Loss: 4329446912.0000
Epoch [71/500], Loss: 4075756288.0000
Epoch [81/500], Loss: 3902465280.0000
Epoch [91/500], Loss: 3761196544.0000
Epoch [101/500], Loss: 3618962176.0000
Epoch [111/500], Loss: 3470758912.0000
Epoch [121/500], Loss: 3310515712.0000
Epoch [131/500], Loss: 3136776960.0000
Epoch [141/500], Loss: 2956176384.0000
Epoch [151/500], Loss: 2779013376.0000
Epoch [161/500], Loss: 2623364608.0000
Epoch [171/500], Loss: 2511877376.0000
Epoch [181/500], Loss: 2453164288.0000
```

MLP训练过程

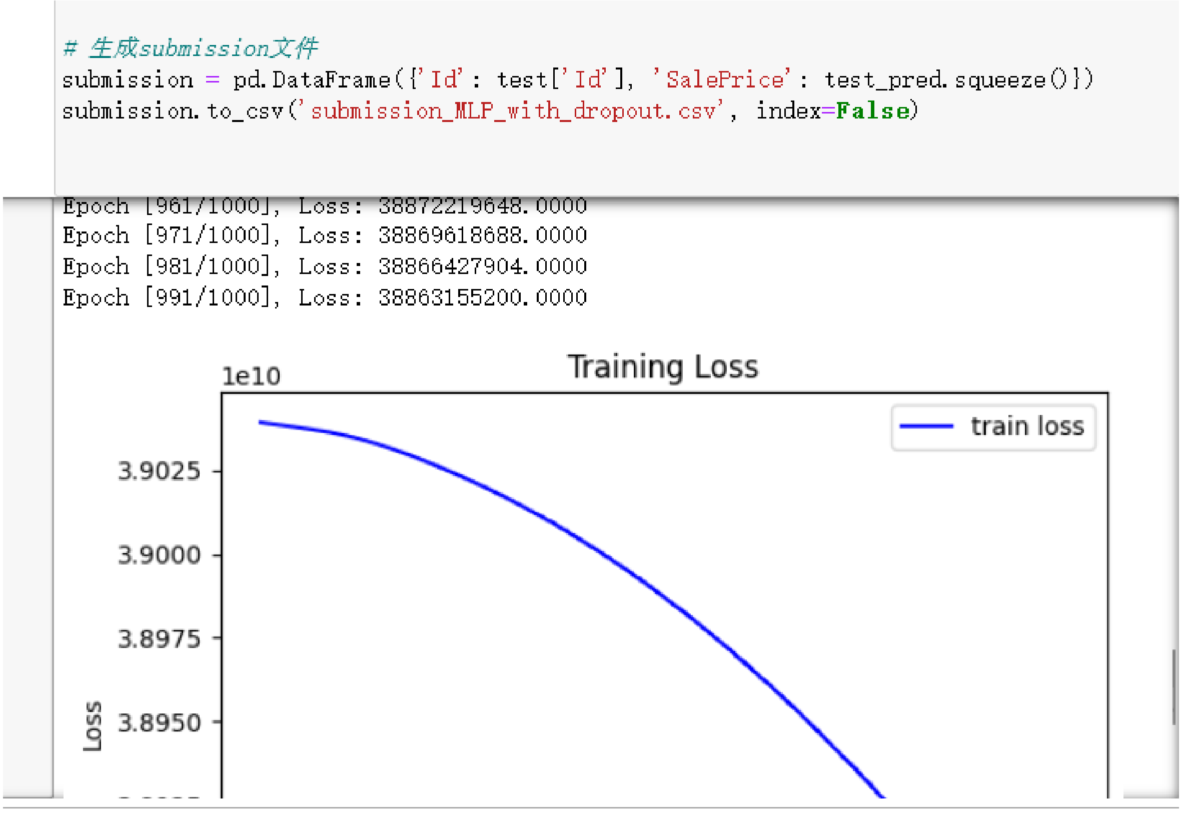


In [42]:

```
# 绘制训练损失曲线
plt.plot(range(num_epochs), train_losses, 'b-', label='train loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

带Dropout的MLP

```
# 设置超参数
input_dim = train_features.shape[1]
output_dim = 1
hidden_dim = 256
lr = 0.001
num_epochs = 1000

# 初始化模型、损失函数以及优化器
model = Net(input_dim, hidden_dim, output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

# 记录训练过程的指标
train_losses = []
# 训练MLP模型
for epoch in range(num_epochs):
    # 前向传播
    outputs = model(train_features)
    # 计算损失
    loss = criterion(outputs, train_labels)
    # 反向传播及优化
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    train_losses.append(loss.item())

    # 每10轮输出一次损失
    if epoch % 10 == 0:
        print('Epoch [{}/{}], Loss:
{:.4f}'.format(epoch + 1, num_epochs, loss.item()))
```

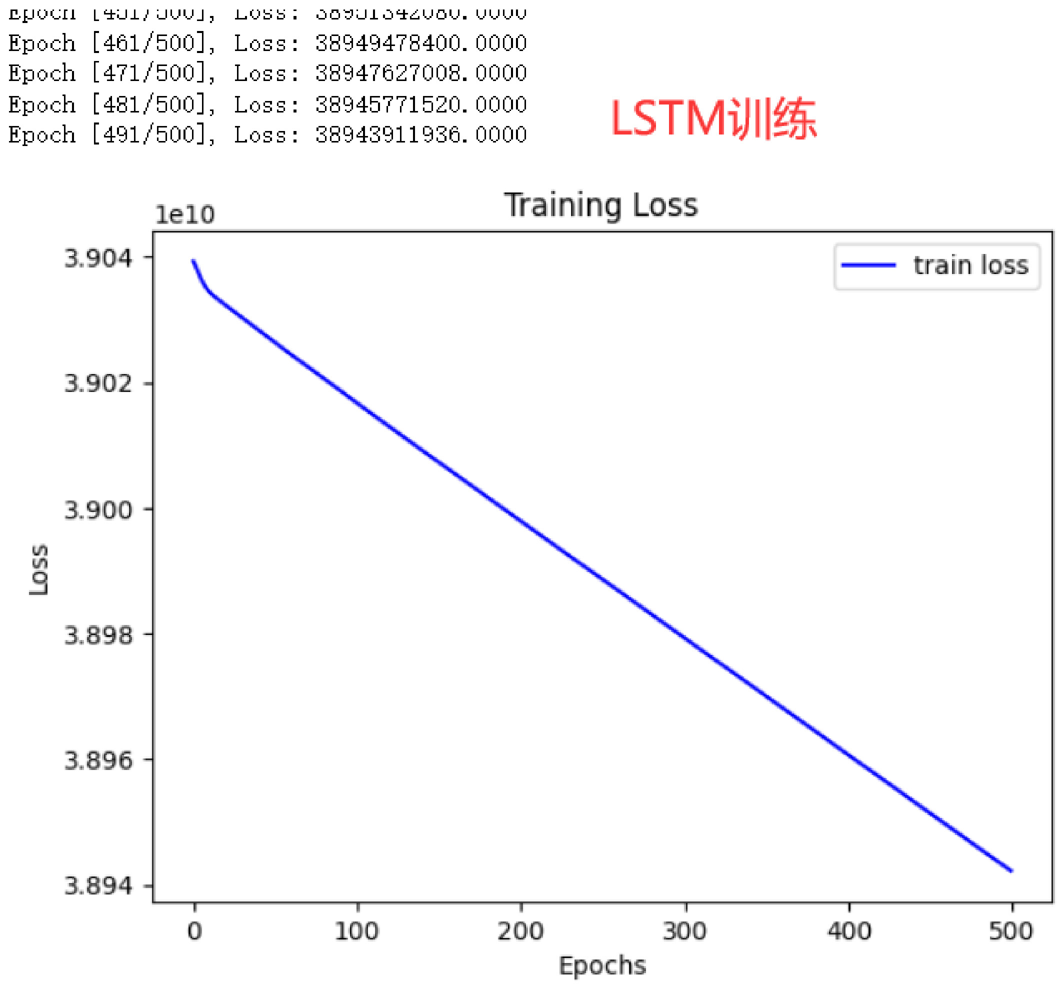


```
# 生成submission文件
submission = pd.DataFrame({'Id': test['Id'], 'SalePrice': test_pred.squeeze()})
submission.to_csv('submission_MLP_with_dropout.csv', index=False)
```

```
Epoch [961/1000], Loss: 38872219648.0000
Epoch [971/1000], Loss: 38869618688.0000
Epoch [981/1000], Loss: 38866427904.0000
Epoch [991/1000], Loss: 38863155200.0000
```

LSTM模型

```
# 设置超参数
input_dim = train_features.shape[1]
output_dim = 1
hidden_dim = 512
num_layers = 2
lr = 0.001
num_epochs = 500

# 初始化模型、损失函数以及优化器
model = Net(input_dim, hidden_dim, num_layers,
output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)
```

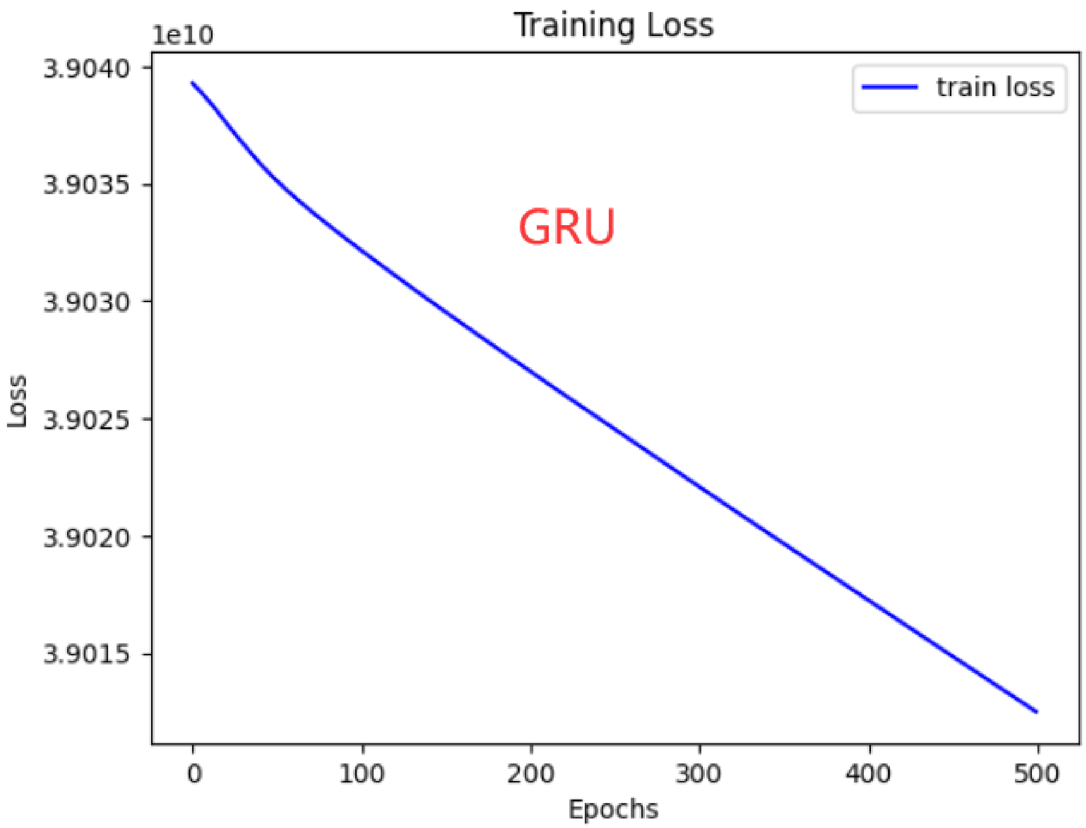


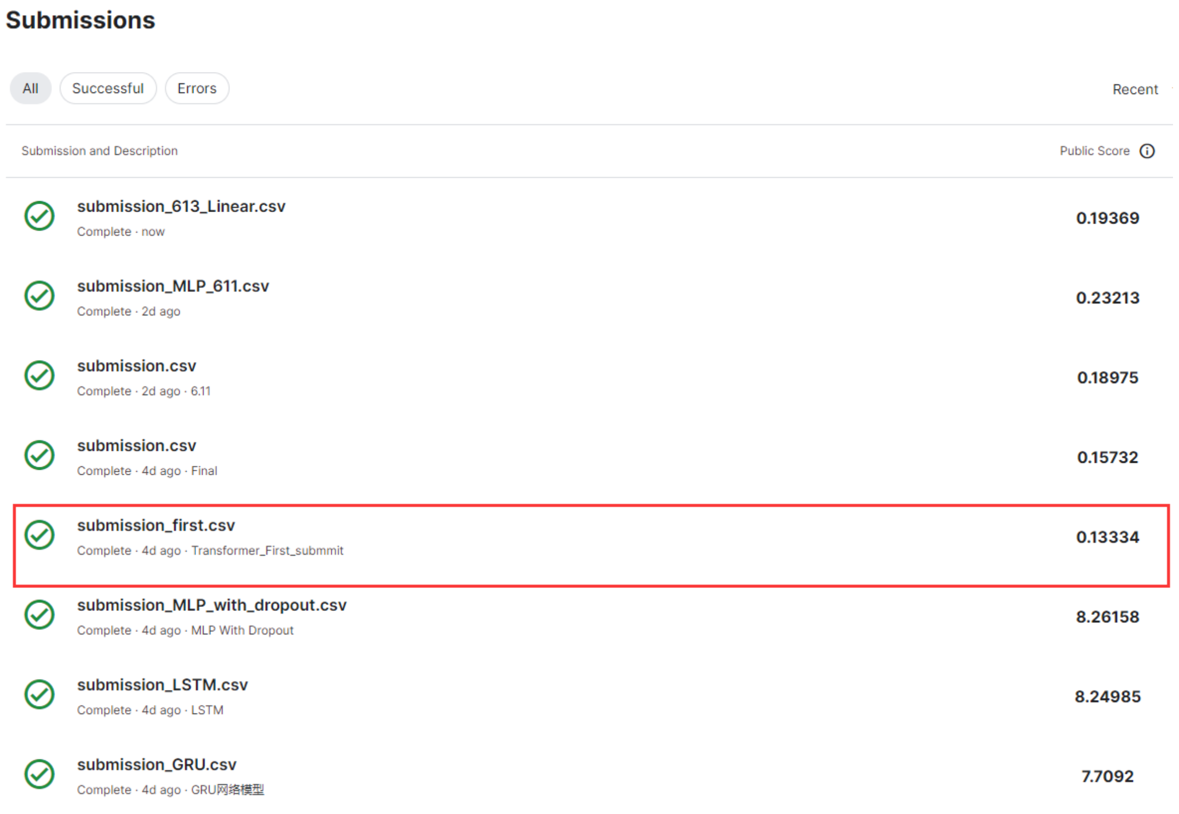
```
Epoch [451/500], Loss: 38951342080.0000
Epoch [461/500], Loss: 38949478400.0000
Epoch [471/500], Loss: 38947627008.0000
Epoch [481/500], Loss: 38945771520.0000
Epoch [491/500], Loss: 38943911936.0000
```

LSTM训练

GRU模型

```
# 定义GRU模型
class Net(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers=2, batch_first=True):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers, batch_first=batch_first)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # 初始化隐层
        h0 = torch.randn(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        # 将隐层传入GRU模型
        out, _ = self.gru(x, h0.detach())
        # 将输出特征传入全连接层
        out = self.fc(out[:, -1, :])
        return out

# 设置超参数
input_dim = train_features.shape[2]
output_dim = 1
hidden_dim = 128
lr = 0.001
num_epochs = 500

# 初始化模型、损失函数以及优化器
model = Net(input_dim, hidden_dim, output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr)
```

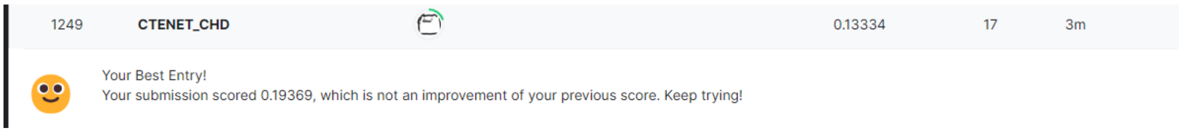




Submissions

All | Successful | Errors | Recent

Submission and Description	Public Score
submission_613_Linear.csv Complete · now	0.19369
submission_MLP_611.csv Complete · 2d ago	0.23213
submission.csv Complete · 2d ago · 6.11	0.18975
submission.csv Complete · 4d ago · Final	0.15732
submission_first.csv Complete · 4d ago · Transformer_First_submmit	0.13334
submission_MLP_with_dropout.csv Complete · 4d ago · MLP With Dropout	8.26158
submission_LSTM.csv Complete · 4d ago · LSTM	8.24985
submission_GRU.csv Complete · 4d ago · GRU网络模型	7.7092

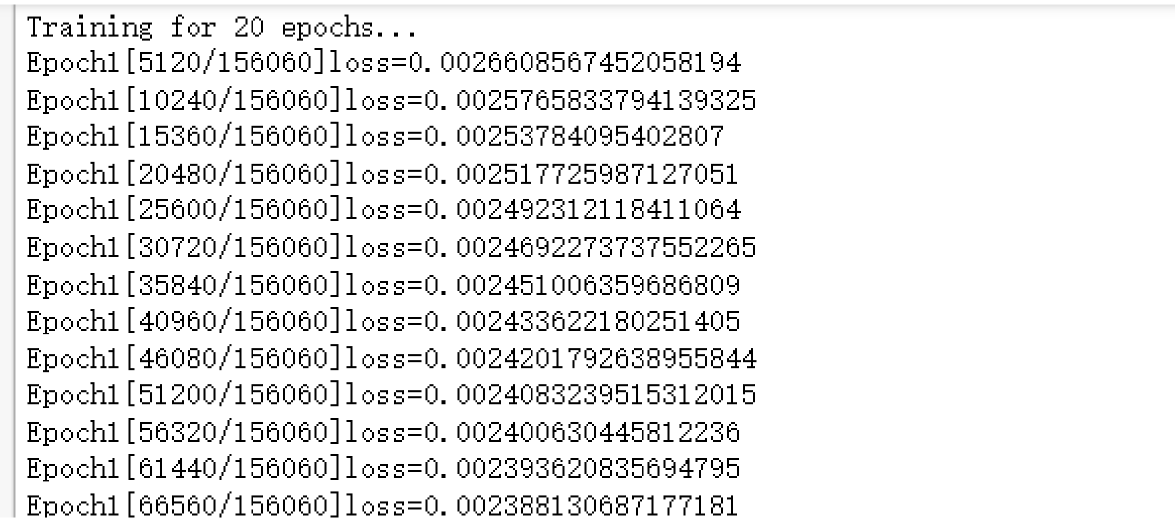


1249	CTENET_CHD	0.13334	17	3m

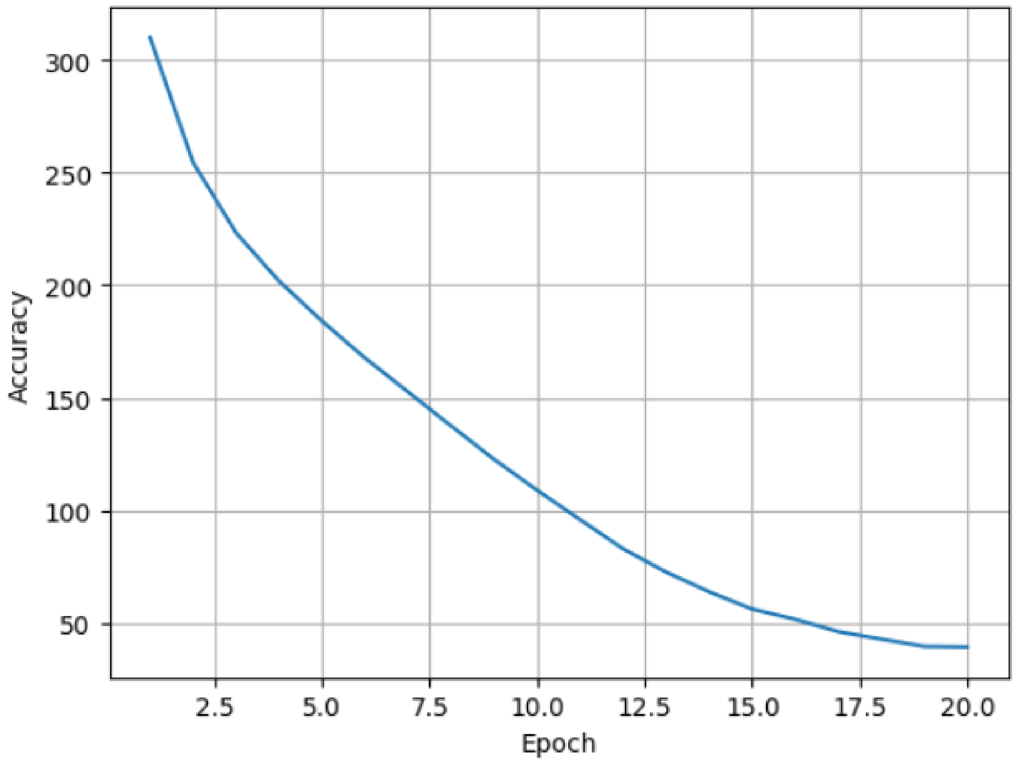
Your Best Entry!
Your submission scored 0.19369, which is not an improvement of your previous score. Keep trying!

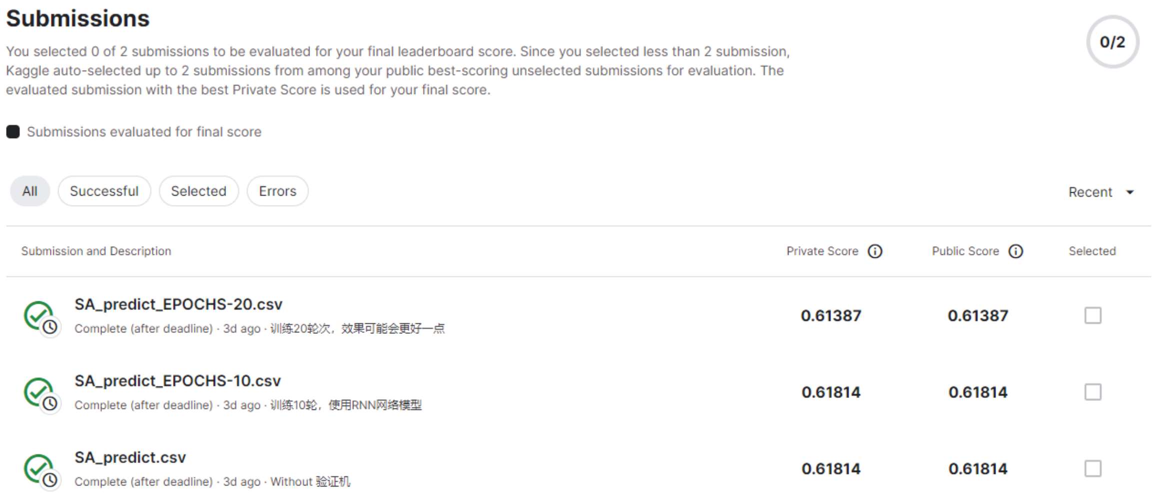
影视评论情感分类训练

```python
#定义训练函数
def train():
    total_loss = 0
    for i, (phrase, sentiment) in enumerate(train_loader, 1):
        inputs, seq_lengths, target = make_tensors(phrase, sentiment)
        output = classifier(inputs, seq_lengths)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        if i % 10 == 0:
            print(f'Epoch{epoch}', end='')
            print(f'[{i * len(inputs)}/{len(train_set)}]', end='')
            print(f'loss={total_loss / (i * len(inputs))}')
    return total_loss
```



```
Training for 20 epochs...
Epoch1[5120/156060]loss=0.0026608567452058194
Epoch1[10240/156060]loss=0.0025765833794139325
Epoch1[15360/156060]loss=0.0025378409540280 7
Epoch1[20480/156060]loss=0.0025177259871270 51
Epoch1[25600/156060]loss=0.0024923121184110 64
Epoch1[30720/156060]loss=0.0024692273737552265
Epoch1[35840/156060]loss=0.0024510063596868 09
Epoch1[40960/156060]loss=0.0024336221802514 05
Epoch1[46080/156060]loss=0.0024201792638955844
Epoch1[51200/156060]loss=0.0024083239515312015
Epoch1[56320/156060]loss=0.0024006304458122 36
Epoch1[61440/156060]loss=0.0023936208356947 95
Epoch1[66560/156060]loss=0.0023881306871771 81
```





Submissions

You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submission, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

0/2

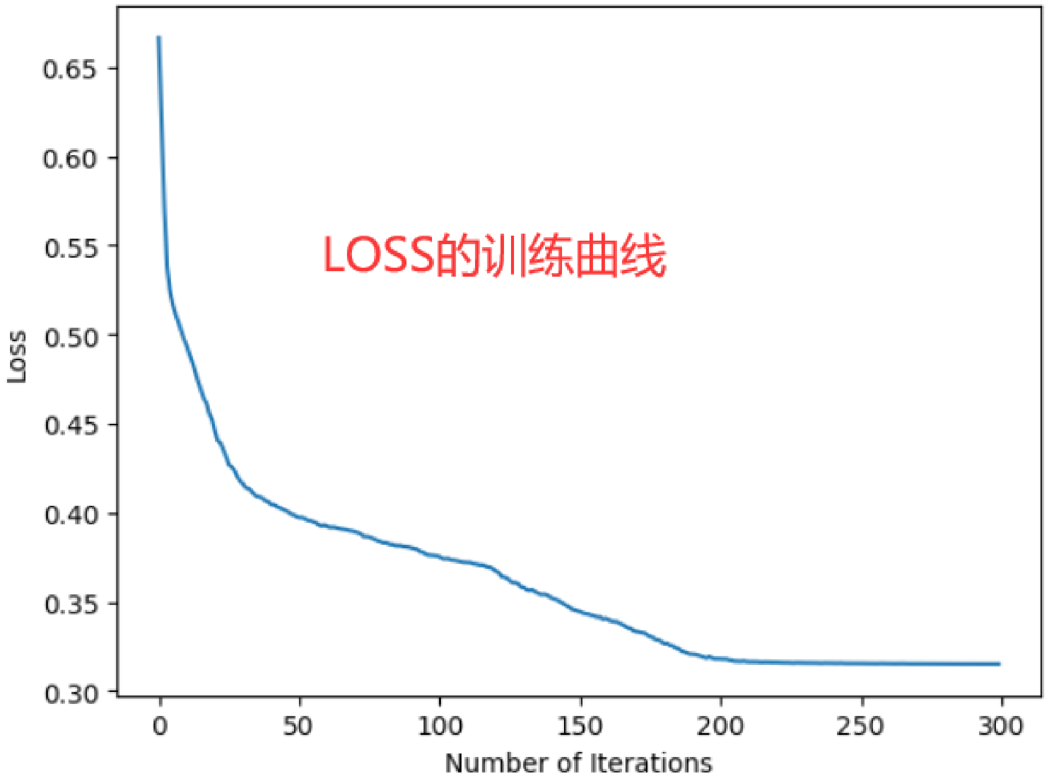
● Submissions evaluated for final score

Submission and Description	Private Score	Public Score	Selected
SA_predict_EPOCHS-20.csv Complete (after deadline) · 3d ago · 训练20轮次，效果可能会更好一点	0.61387	0.61387	□
SA_predict_EPOCHS-10.csv Complete (after deadline) · 3d ago · 训练10轮，使用RNN网络模型	0.61814	0.61814	□
SA_predict.csv Complete (after deadline) · 3d ago · Without 验证机	0.61814	0.61814	□

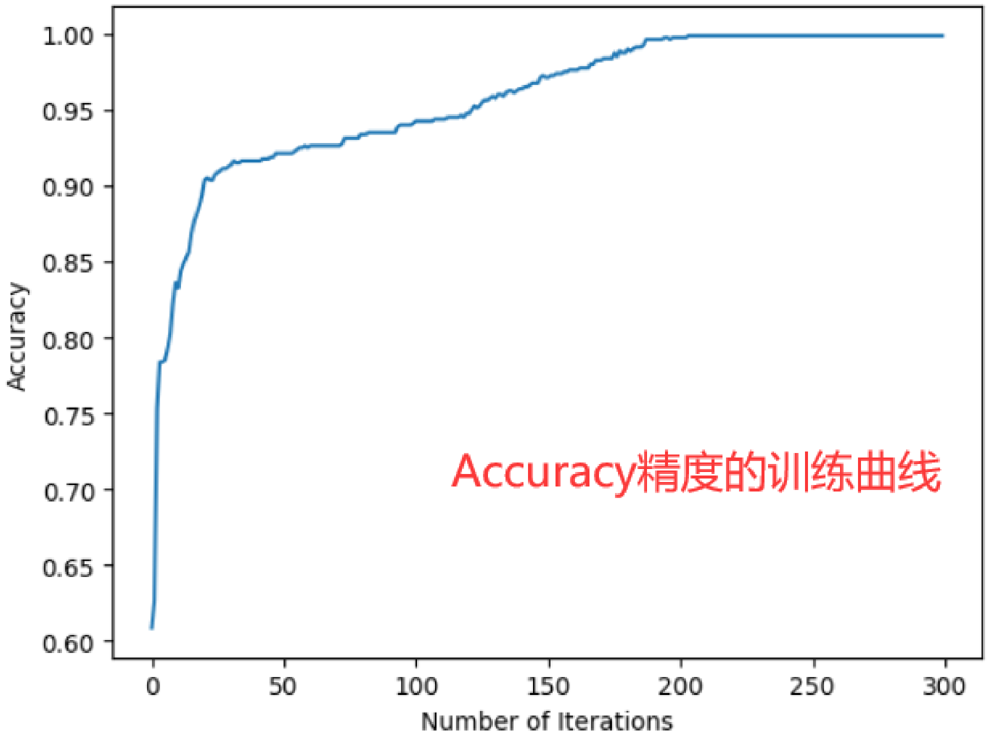
泰坦尼克号存活率预测问题

```
epochs=10
loss_fn = nn.BCELoss().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.01)
for epoch in range(epochs + 1):
    for batch_idx, samples in
enumerate(train_dataset):
        x_train, y_train = samples
        optimizer.zero_grad()
        prediction = model(x_train)
        cost = loss_fn(prediction, y_train)
        cost.backward()
        optimizer.step()

        if batch_idx%250 == 0:
            print('Epoch {:4d}/{} Batch {}/{} Cost:
{:.6f}'.format(
                epoch, epochs, batch_idx+1,
len(train_dataset),
                cost.item()
                ))
    validation_data_eval = []
    for batch_idx, samples in
enumerate(val_dataset):
        x_train, y_train = samples
        prediction = model(x_train)
        cost = loss_fn(prediction, y_train)
        validation_data_eval.append(cost.item())
    print("validation cost : ",
np.mean(validation_data_eval))
```

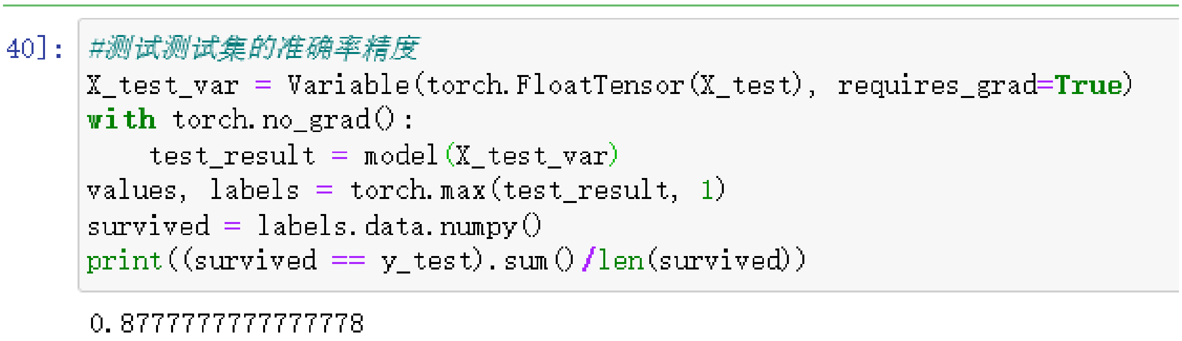
```
input_dim = 1730
output_dim = 2
learning_rate = 1
model = LinearRegression(input_dim,output_dim)
error = nn.CrossEntropyLoss()  #交叉熵损失
optimizer = torch.optim.SGD(model.parameters(),
lr=learning_rate, momentum = 0.5)

for iteration in range(iteration_number):
    batch_loss = 0
    batch_accur = 0
    temp = 0

    for (x, y) in generate_batches(X_train,
y_train, batch_size):
        inputs =
Variable(torch.from_numpy(x)).float()
        labels = Variable(torch.from_numpy(y))
        optimizer.zero_grad()

        results = model(inputs)

        loss = error(results, labels)

        batch_loss += loss.data

        loss.backward()

        optimizer.step()

        with torch.no_grad():
            _, pred = torch.max(results, 1)
            batch_accur += torch.sum(pred ==
labels)
            temp += len(pred)

    loss_list.append(batch_loss/batch_no)
    acc_list.append(batch_accur/temp)

    if(iteration % 50 == 0):
        print('epoch {}: loss {}, accuracy
{}'.format(iteration, batch_loss/batch_no,
batch_accur/temp))
```



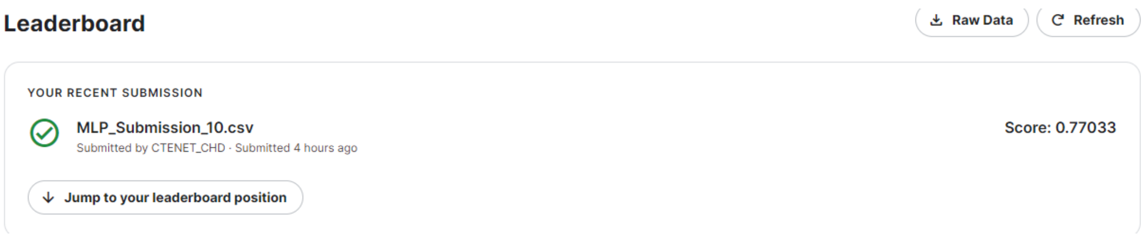
LOSS的训练曲线



Accuracy精度的训练曲线



```
40]:  #测试测试集的准确率精度
      X_test_var = Variable(torch.FloatTensor(X_test), requires_grad=True)
      with torch.no_grad():
          test_result = model(X_test_var)
      values, labels = torch.max(test_result, 1)
      survived = labels.data.numpy()
      print((survived == y_test).sum()/len(survived))

      0.8777777777777778
```

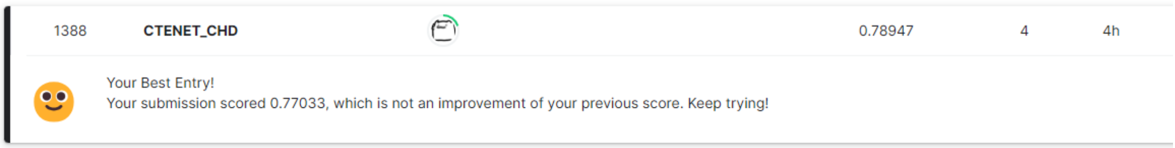


Leaderboard

YOUR RECENT SUBMISSION

MLP_Submission_10.csv
Submitted by CTENET_CHD · Submitted 4 hours ago

Score: 0.77033

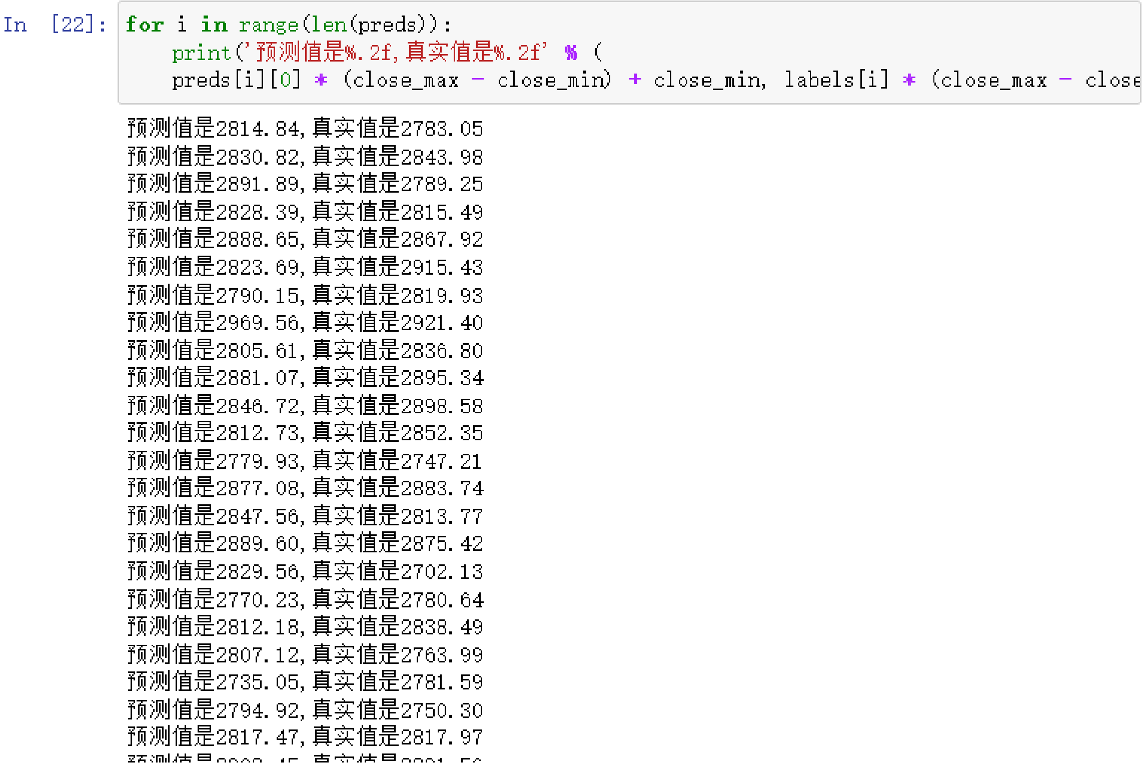


1388	CTENET_CHD	0.78947	4	4h

Your Best Entry!
Your submission scored 0.77033, which is not an improvement of your previous score. Keep trying!

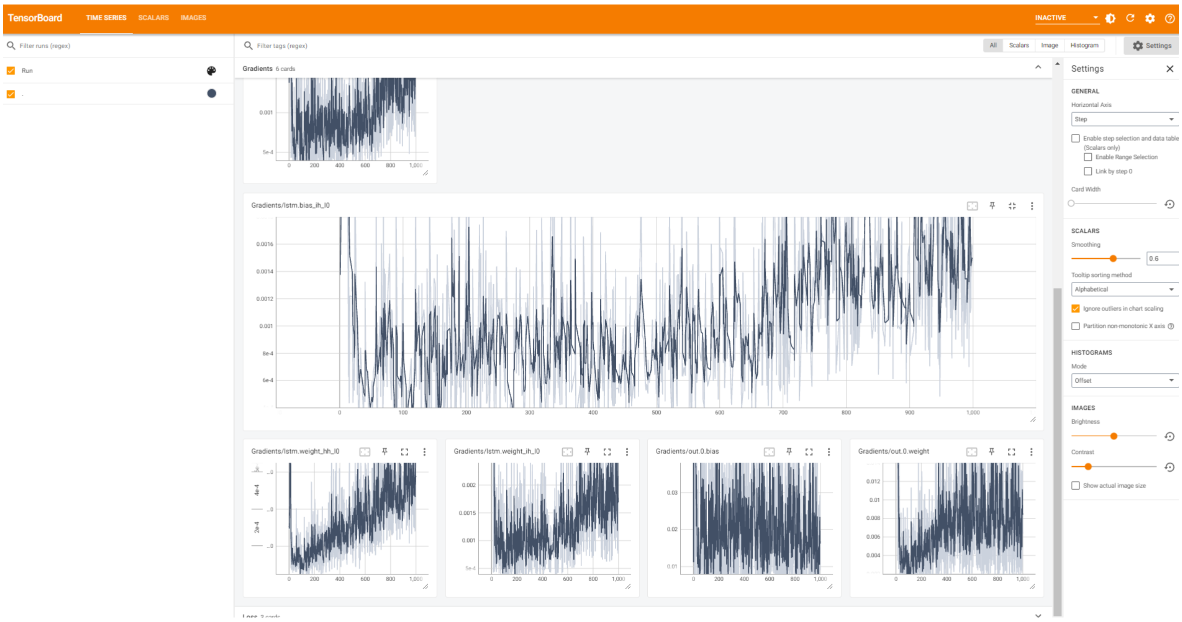
股票问题

```python
# 指定保存日志的路径和名称
writer = SummaryWriter(log_dir='./logs')
for i in range(epochs):
    total_loss = 0
    for idx, (data, label) in enumerate(train_loader):
        if useGPU:
            data1 = data.squeeze(1).cuda()
            pred = model(Variable(data1).cuda())
            # print(pred.shape)
            pred = pred[1,:,:]
            label = label.unsqueeze(1).cuda()
            # print(label.shape)
        else:
            data1 = data.squeeze(1)
            pred = model(Variable(data1))
            pred = pred[1, :, :]
            label = label.unsqueeze(1)
        loss = criterion(pred, label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    mean_loss = total_loss / len(train_loader.dataset)
    writer.add_scalar('Train/Loss', mean_loss, i)
    print(total_loss)
    if i % 10 == 0:
        # torch.save(model, args.save_file)
        torch.save({'state_dict':
model.state_dict()}, './weights/stock.pkl')
        print('第%d epoch, 保存模型' % i)
writer.close()
# torch.save(model, args.save_file)
torch.save({'state_dict': model.state_dict()},
'./weights/stock.pkl')
```



```
In  [22]: for i in range(len(preds)):
              print('预测值是%.2f,真实值是%.2f' % (
              preds[i][0] * (close_max - close_min) + close_min, labels[i] * (close_max - close
```

```
预测值是2814.84,真实值是2783.05
预测值是2830.82,真实值是2843.98
预测值是2891.89,真实值是2789.25
预测值是2828.39,真实值是2815.49
预测值是2888.65,真实值是2867.92
预测值是2823.69,真实值是2915.43
预测值是2790.15,真实值是2819.93
预测值是2969.56,真实值是2921.40
预测值是2805.61,真实值是2836.80
预测值是2881.07,真实值是2895.34
预测值是2846.72,真实值是2898.58
预测值是2812.73,真实值是2852.35
预测值是2779.93,真实值是2747.21
预测值是2877.08,真实值是2883.74
预测值是2847.56,真实值是2813.77
预测值是2889.60,真实值是2875.42
预测值是2829.56,真实值是2702.13
预测值是2770.23,真实值是2780.64
预测值是2812.18,真实值是2838.49
预测值是2807.12,真实值是2763.99
预测值是2735.05,真实值是2781.59
预测值是2794.92,真实值是2750.30
预测值是2817.47,真实值是2817.97
```



Settings

GENERAL

Horizontal Axis: Step

□ Enable step selection and data table (Scalars only)

□ Enable Range Selection

□ Link by step 0

Card Width

SCALARS

Smoothing: 0.6

Tooltip sorting method: Alphabetical

☑ Ignore outliers in chart scaling

□ Partition non-monotonic X axis

HISTOGRAMS

Mode: Offset

IMAGES

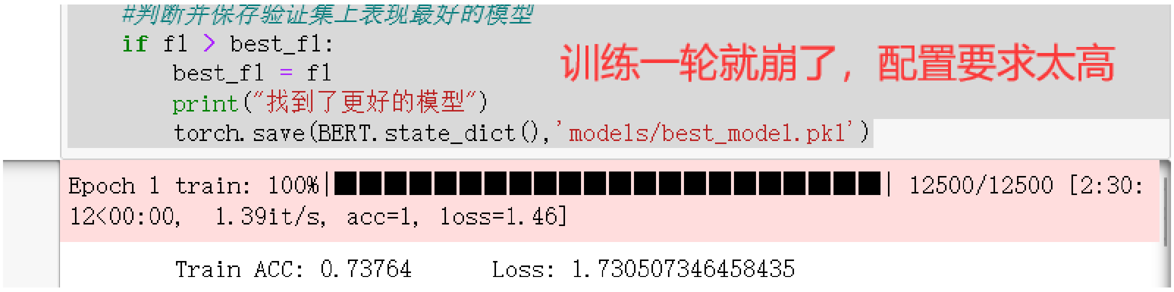
Brightness

Contrast

□ Show actual image size

BERT+Transformer+BiLSTM

```
## 设置预训练超参数
batch_size = 4
device = 'cuda' if torch.cuda.is_available() else 'cpu'
epochs = 10   # 训练轮次
learning_rate = 5e-6   #学习率设置的比较低


for epoch in range(1,epochs+1):
    losses = 0  #损失
    accuracy = 0  # 准确率
    BERT.train()   #训练
    train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    train_bar = tqdm(train_dataloader, ncols=100)
    for input_ids,token_type_ids,attention_mask,label_id in train_bar:
        #梯度清零
        BERT.zero_grad()
        train_bar.set_description('Epoch %i train' % epoch)

        #传入数据 调用 model.forward()
        output = BERT(
            input_ids=input_ids.to(device),
            attention_mask=attention_mask.to(device),
            token_type_ids=token_type_ids.to(device))

        #计算loss
        loss= criterion(output,label_id.to(device))
        losses += loss.item()

        pred_labels = torch.argmax(output,dim=1)  # 预测的label
        acc = torch.sum(pred_labels ==
label_id.to(device)).item() / len(pred_labels) #acc
        accuracy += acc

        loss.backward()
        optimizer.step()
        train_bar.set_postfix(loss =
loss.item(),acc=acc)
    average_loss = losses / len(train_dataloader)
    average_acc = accuracy / len(train_dataloader)

    print('\tTrain ACC:', average_acc, '\tLoss:',
average_loss)

    # 保存训练集的loss和accuracy供后续可视化
    train_losses.append(average_loss)
    train_accs.append(average_acc)

    # 验证
    model.eval()
    losses = 0  # 损失
    pred_labels = []
    true_labels = []
    valid_bar = tqdm(valid_dataloader, ncols=100)
    for input_ids, token_type_ids, attention_mask,
label_id in valid_bar:
        valid_bar.set_description('Epoch %i valid'
% epoch)

        output = model(
            input_ids=input_ids.to(device),

 attention_mask=attention_mask.to(device),

 token_type_ids=token_type_ids.to(device),
        )

        loss = criterion(output,
label_id.to(device))
        losses += loss.item()
        pred_label = torch.argmax(output, dim=1)  #
预测出的label
        acc = torch.sum(pred_label ==
label_id.to(device)).item() / len(pred_label)  #
acc
        valid_bar.set_postfix(loss=loss.item(),
acc=acc)


 pred_labels.extend(pred_label.cpu().numpy().tolist
())

 true_labels.extend(label_id.numpy().tolist())

    average_loss = losses / len(valid_dataloader)
    print('\tLoss:', average_loss)
    # 保存验证集的loss供后续可视化
    valid_losses.append(average_loss)

    #分类报告
    report =
metrics.classification_report(true_labels,
pred_labels, labels=valid_dataset.labels_id,

target_names=valid_dataset.labels)
    print('* Classification Report:')
    print(report)

    # f1 用来判断最优模型
    f1 = metrics.f1_score(true_labels, pred_labels,
labels=valid_dataset.labels_id, average='micro')

    if not os.path.exists('models'):
        os.makedirs('models')

    #判断并保存验证集上表现最好的模型
    if f1 > best_f1:
        best_f1 = f1
        print("找到了更好的模型")
 torch.save(BERT.state_dict(),'models/best_model.pkl')
```



```
    #判断并保存验证集上表现最好的模型
    if f1 > best_f1:
        best_f1 = f1
        print("找到了更好的模型")
        torch.save(BERT.state_dict(),'models/best_model.pkl')
```

训练一轮就崩了，配置要求太高

```
Epoch 1 train: 100%|██████████████████████| 12500/12500 [2:30:12<00:00,  1.39it/s, acc=1, loss=1.46]

        Train ACC: 0.73764      Loss: 1.730507346458435
```

```
# 设置超参数
input_dim = train_features.shape[1]
output_dim = 1
hidden_dim = 512
num_layers = 2
lr = 0.001
num_epochs = 500

# 初始化模型、损失函数以及优化器
model = Net(input_dim, hidden_dim, num_layers, output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)
```