量化投资作业 2

Author's Name：黄倪远
ID：3200101028

1. “04_train_model_nn_static.py”主程序中，定义模型时设置了一系列超参数（d_feat、hidden_size等），叙述这些超参数的含义和作用。

- `d_feat`：因子数。
- `hidden_size`：隐状态数。隐藏层中节点的个数
- `n_epochs`：迭代次数。前向和反向传播中所有批次的单次训练迭代，简单来说就是数据被迭代的次数。
- `lr`：学习率。是用来指导如何用损失函数的梯度调整网络权重的超参数。学习率过大可能会越过最优点，学习率过小会使损失函数变化的速率很慢，增加网络的收敛的复杂度同时可能会困在局部鞍点。
- `early_stop`：早停次数。是一种交叉策略，把一部分训练集保留作为验证集，当看到验证集上的性能变差时，就立即停止模型的训练。选择合适的早停次数可以防止过拟合和欠拟合的出现。
- `optimizer`：优化器。优化器就是在深度学习反向传播过程中，指引损失函数（目标函数）的各个参数往正确的方向更新合适的大小，使得更新后的各个参数让损失函数（目标函数）值不断逼近全局最小。源代码中用的Adam是用SGD的一阶动量加上AdaDelta的二阶动量，有效控制学习率增长步长和梯度方向，防止梯度的震荡和在鞍点的静止。
- `loss`：损失函数，用来评价模型的预测值和真实值不一样的程度。源代码中用均方误作为损失函数。
- `GPU`：GPU编号。
- `n_jobs`：最大并发进程数。
- `seed`：随机数种子。
- `save_path`：模型存储路径。

2. 针对“04_train_model_nn_static.py”进行扩展，测试下列关键超参数的不同取值对模型的影响，并分析原因：

分别记录`hidden_size`和`lr`取不同值时的best score。其中仅改变相应的超参数，其余超参数都保持源样例。

2.1 hidden_size

hidden_size	best epoch	best score
16	35	-0.983564
36	11	-0.982095
64（原始值）	17	-0.980471
81	17	-0.978347
100	17	-0.979518
144	17	-0.981205
169	17	-0.980588
196	17	-0.982907
225	1	-0.983492

2.2 lr

lr	best epoch	best score
0.000001	76	-0.987084
0.00001	49	-0.985043
0.0001（原始值）	17	-0.980471
0.001	4	-0.983001
0.01	7	-0.985949
0.1	0	-0.984158

2.3 分析

实验结果可以发现，只有选择合适的`hidden_size`和`lr`才能得到较好的结果，过大或者过小都会对学习过程造成负面影响。

`hidden_size`的值过小会导致模型的拟合效果较差，随着`hidden_size`值的增加，模型的拟合效果变好，但训练的时间也会增大，最重要的是会有过拟合的问题，像实验中`hidden_size`到225的时候，第2次训练就已经是最好的效果了。

`lr`是学习的步长，`lr`过下会导致学习过慢，最重要的是有可能会被困在局部最小点。`lr`过大则有可能跳过最优解。

3. “model/pytorch_nn.py”是核心模型方法，其中神经网络模型在NNModel类中定义，叙述网络结构，网络总共有多少自由参数？每层各有多少自由参数？

- 神经网络结构如下：
  - 输入层：d_feat（=10）个输入单元
  - 隐藏层：每层都是hidden_size（=64）个单元
    - bn1：
      $d\_feat \times (hidden\_size \times hidden\_size) \times hidden\_size + hidden\_size$
      2,621,504个自由参数
    - bn2：
      $hidden\_size \times (hidden\_size \times hidden\_size) \times hidden\_size + hidden\_size$
      16,777,280个自由参数
    - fc： $hidden\_size \times (hidden\_size \times hidden\_size) \times 1 + 1$ 262,145个自由参数
  - 输出层：1个输出单元
- 总共自由参数个数：19,660,929

4. 针对“model/pytorch_nn.py”进行扩展，将损失函数改为以下值，考察这些损失函数下，模型IC值表现：

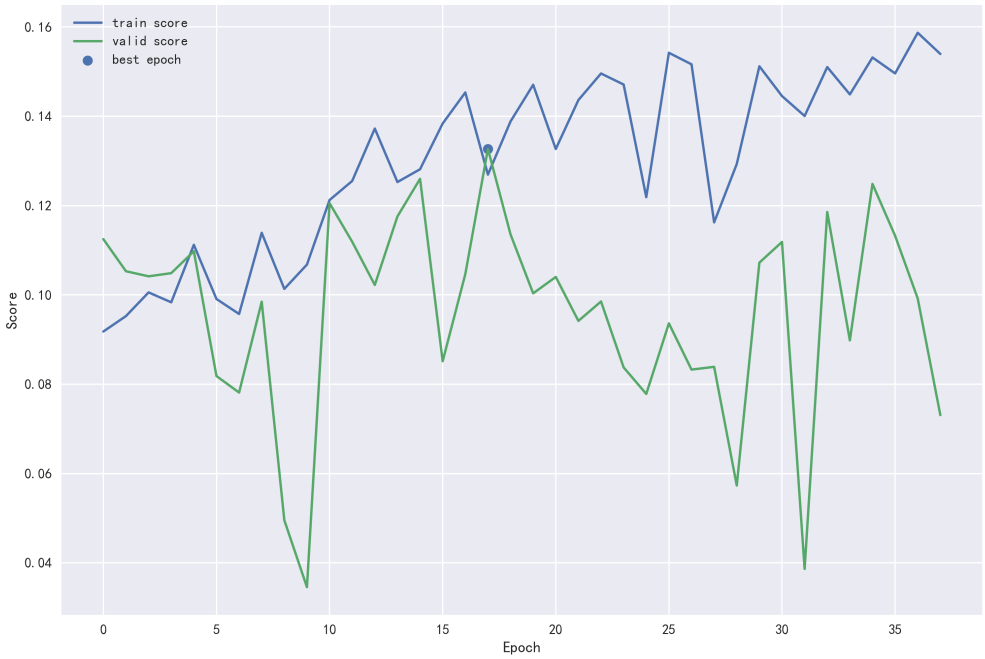
```
    def loss_fn(self, pred, label):
        mask = ~torch.isnan(label)

        if self.loss == "mse":
            return self.mse(pred[mask], label[mask])
        if self.loss == "IC":
            return self.IC(pred[mask], label[mask])
        if self.loss == "weighted_mse":
            return self.weighted_mse_loss(pred[mask], label[mask])

        raise ValueError("unknown loss `%s`" % self.loss)
```

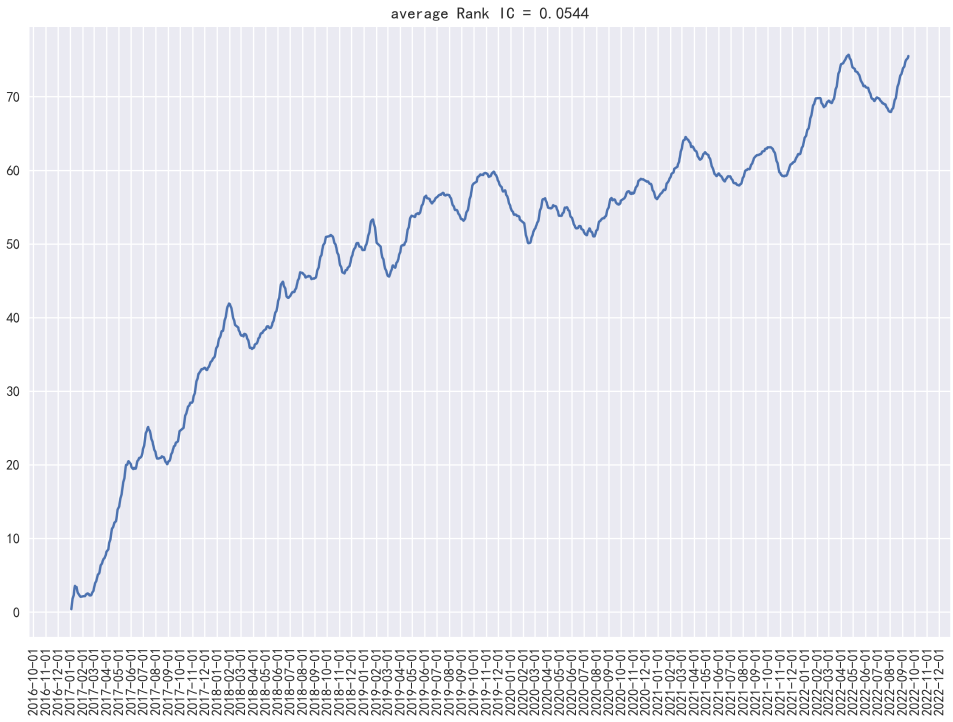
4.1 IC值。

- 损失函数改成：

```
def IC(self, pred, label):
    v_pred = pred - torch.mean(pred)
    v_label = label - torch.mean(label)
    loss =
torch.mean(torch.sum(v_label*v_pred)/(torch.sqrt(torch.sum(v_label**2))
* torch.sqrt(torch.sum(v_pred**2))))
    return loss
```

- 结果：



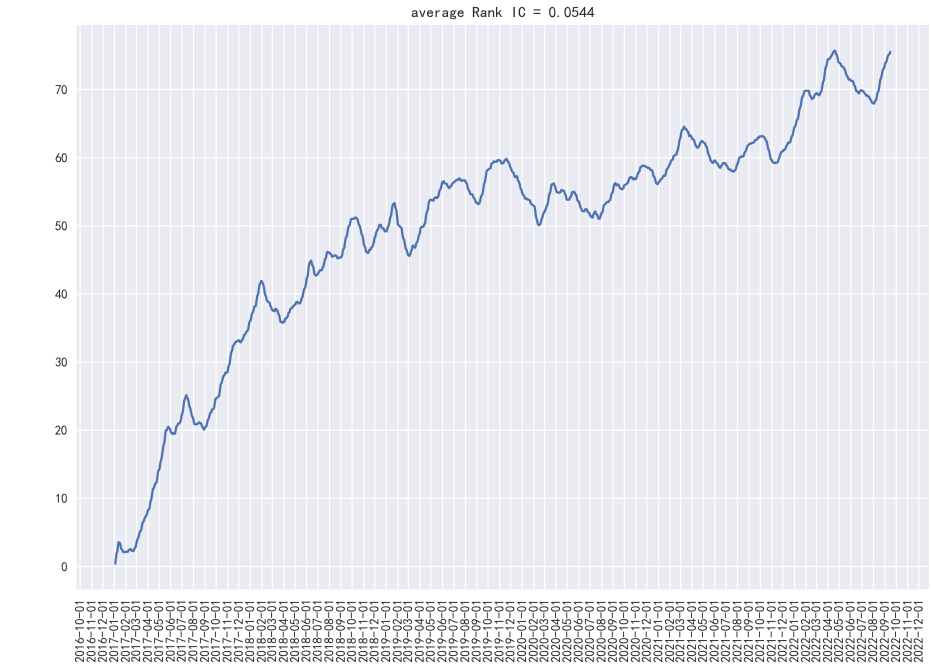


4.2 上节课介绍的半衰加权**MSE**。每个截面对股票收益排序，高收益股票给更高权重，半衰期可设为**0.5**，即收益排名第一的股票权重为**1**，排名中位数的股票权重为**0.5**，排名后**25%**分位数的股票权重为**0.25**。

- 损失函数为：

```
    def weighted_mse_loss(self, pred, label):
        observation_dim = pred.size()[-1]
        mid_dim = int(0.5*observation_dim)
        midmid_dim = int(0.75*observation_dim)
        sort_pred, index_pred = torch.sort(pred,0,descending=True)
        sort_label, index_label = torch.sort(label, 0,
descending=True)
        total = 0
        for i in range(observation_dim):
            if i<mid_dim:
                total = total + (sort_pred[i]-sort_label[i])**2
            elif i<midmid_dim:
                total = total + (sort_pred[i]-sort_label[i])**2 *
0.5
            else:
                total = total + (sort_pred[i] - sort_label[i]) **
2 * 0.25
        loss = torch.mean(total/observation_dim)
        return loss
```

- 结果



4.3 上节课介绍的**logistic**有序回归，类别数可设为**5**。

（这部分未能实现，不理解为什么有序回归可以作为损失函数）