# 基于深度强化学习方法的沪深 300 指数择时策略研究

2024 UIBE 硕士课程：Python 与金融量化

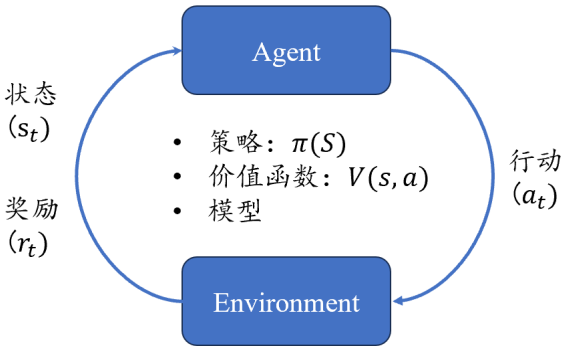
任中兴

202420210625

## 报告概要

本文采用基于深度强化学习（DQN）的方法进行股指日频择时策略。强化学习方法通过智能体与环境交互的方式进行学习，采取不同的行动获取不同的结果，并从中学习如何更好的适应环境。针对长期目标的试错学习，是强化学习的特殊之处。有别于现有的其他机器学习方法通过监督学习的方式对真实标签进行拟合，强化学习的思想更接近人类的真实决策过程。强化学习不存在标准答案，而是这对长期目标的试错学习，在数学上使用马尔可夫决策过程进行刻画。

强化学习的核心是智能体通过与环境的交互，从信号反馈中进行学习。智能体首先观察环境的状态($State$)，并依次为依据采取某种动作(Action)，该动作会对后续的环境产生影响($State_{t+1}$)。随后，环境下一刻的状态和该动作产生的奖励将反馈给智能体。训练智能体的目标是尽可能多地从环境里获取总奖励($Reward$)。总奖励并不是下一时刻的即使奖励，而是未来每一时刻的奖励“折现”之和，这一点与金融投资的理念不谋而合。通过尝试和相对简单的反馈循环，学习最佳策略。下图展示了一个简单的强化学习模型组件的构成：



本项目则基于历史行情数据训练 DQN 模型，运用多组随机数种子合成信号，基于测试集进行日频调仓回测。以上证指数为择时标的，2008 至 2018 年为训练集，2019-2023 年为测试集，交易费率双边 0.25%，原始超参数测试集年化收益率达到 5.34%，夏普比率 0.5。

强化学习对随机数设定和超参数敏感。本文采取模型平均和信号合成的方式尽量控制随机数对模型结果的影响。同时本文考察折扣因子、回放内存、回看区间、预测区间等超参数影响，超参数调整会较大程度影响模型最终表现。强化学习泛化能力在金融领域受限，部分研究则错误的将测试集也归入到训练集中，在本文中也注意到并修正了这一点，修正后样本外回测表现基本趋于正常水平。

## 强化学习原理

### 策略

策略是一种算法或一组规则，用于描述智能体的决策方式。通常策略用函数$\pi$表示，它将状态（s）映射到行动（a）：

$$\mathrm{a}_t = \pi(s_t) \tag{1}$$

智能体依据给定当前状态做出行动决策。策略或确定或随机。确定策略将一种状态映射到行动。反之，随机策略则输出行动的概率分布，也就是说，智能体并不直接确定执行某一行动，而是为给定状态下的各种行动分配执行概率。在强化学习的任务中，我们的目标是学习最优策略（记作$\pi^*$）。最优策略意味着最大化每隔状态的收益。

### 价值函数

上文所述的强化学习“收益”是最大化未来奖赏或累积折扣奖赏 G。定义为不同时刻奖励函数 R 的函数。

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

其中折扣因子$\gamma$取值范围是$(0,1)$，这一概念与金融学中货币的时间价值的思想一致：即用来“惩罚”未来的奖赏，面对不能直接提供好处同时还要承担不确

定性的未来奖励，需要对其折现来表示对当前时刻的效用。价值函数通过预测未来奖赏 G，度量一种状态的潜在价值：

$$\mathrm{V(s)} = E[G_t \mid S_t = s] \tag{3}$$

类似的，定义“状态-行动”对“行动-价值”函数为：

$$\mathrm{Q}(s,a) = E[G_t \mid S_t = s, A_t = a] \tag{4}$$

公式中的期望消除了 t 时刻之后的所有状态$S_{t+1}, \dots, S_n$与动作$A_{t+1}, \dots, A_n$。**最优动作价值函数**用最大化消除策略$\pi^*$：

$$Q_\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t), \forall s_t \in S, a_t \in A \tag{5}$$

$Q_*$是最优动作价值，即已知$s_t$和$a_t$，无论未来采取何种策略$\pi$，回报$G_t$的期望都不可能超过$Q_*$。

在实践中，我们训练 DQN 预测$Q(s,a;\theta)$使之尽可能接近$Q_*$。下面将介绍一种用于训练 DQN 的计算方法：TD 算法。根据公式(2)可知回报$G_t$和$G_{t+1}$之间的关系是：

$$\mathrm{G}_t = \mathrm{R}_t + \gamma \cdot \underbrace{\sum_{k=t+1}^{n} \gamma^{k-t-1} \cdot \mathrm{R}_k}_{=G_{t+1}} \tag{6}$$

根据公式(5)(6)，我们可以得到一个定理，被称为**最优贝尔曼方程**：

$$\underbrace{\mathrm{Q}_\star(s_t, a_t)}_{\mathrm{G}_t\text{的期望}} = \mathbb{E}_{S_{t+1} \sim p(\cdot|s_t, a_t)}[R_t + \gamma \cdot \underbrace{\max_{A \in \mathcal{A}} Q_\star(S_{t+1}, A)}_{G_{t+1}\text{的期望}} \mid S_t = s_t, A_t = a_t] \tag{7}$$

贝尔曼方程的右边定义成一个期望，当智能体执行动作$a_t$之后，环境通过转态转移函数$p(s_{t+1}|s_t, a_t)$计算出新的转态$s_{t+1}$。奖励$R_t$最多只依赖于$S_t$、$A_t$、$S_{t+1}$。当我们观测到$S_t$、$A_t$、$S_{t+1}$时，奖励$R_t$也会被观测到，这样就可以得到一个四元组：

$$(s_t, a_t, r_t, s_{t+1}) \tag{8}$$

根据上式中的四元组我们可以计算出：

$$\mathrm{r}_t + \gamma \cdot \max_{a \in A} Q_\star(s_{t+1}, a) \tag{9}$$

公式(9)是公式(7)右端期望的蒙特卡洛近似，因此我们可以认为：

$$\mathrm{Q}_{\star}(s_t, a_t) \approx \mathrm{r}_t + \gamma \cdot \max_{a \in A} Q_{\star}(s_{t+1}, a) \tag{10}$$

公式(10)左边是神经网络在$t$时刻做出的预测。右边是神经网络在$t+1$时刻做出的预测，它部分基于事实观测到的奖励$r_t$。虽然二者都是对最优价值函数$Q_*$的估计，但是右端基于部分实施所以更加可信。在这里我们定义神经网络的损失函数使左端预测值逼近右端来进行神经网络参数的训练。

在这里值得注意的是，DQN 训练的过程中与智能体的策略$\pi$无关。这就意味着实质上可以用任何策略控制智能体与环境交互。然而这为了使智能体学习和做出最优决策，这里使用**$\epsilon$-贪心算法($\epsilon$-greedy)**作为策略函数去控制智能体与环境的交互。该方法以概率$\epsilon$随机选择一种行动，或根据 Q 值函数，以$1-\epsilon$的概率选择最佳行动：

$$a_t = \begin{cases} \arg\max_a Q(s_t, a; \theta), (1-\epsilon) \\ \text{random select a} \in \text{A}, (\epsilon) \end{cases} \tag{11}$$

## 日频择时策略的构建

### 状态空间 S

表征状态的原始数据为股指在回测区间内的日度开盘价、收盘价、最高价、最低价和成交量。预处理方式为计算$t-lookback+1$至$t$日的行情数据相较于过去 252 个交易日收盘价的 Z 分数。Z 值标准化的方式如下所示。因此状态空间 S 为$lookback*5$维空间。回看区间去 5 个交易日，同时测试 10 和 15。

$$Z = (X - \mu_X) / \sigma_X \tag{12}$$

### 动作空间 A

动作空间 A 的定义为：$\text{A} = \{buy, sell, hold\}$。其中 buy 表示全仓做多，sell 表示平仓，hold 表示持有多仓或者保持空仓，不涉及做空。基于 t 日的状态空间 S 做出选择，以 t+1 日的开盘价执行交易。

### 状态转移矩阵 P

在本项目中，我们使用无模型的 DQN 方法进行状态和行动的交互。无模型方法不需要状态转移矩阵，智能体日通过与环境互动选择进入下一状态。

**奖励 R**

在本项目中，奖励分为四种情况：

- 当前未持仓，且$A_t$为 buy 时，奖励为预测区间内扣费后的多头收益率：

$$\mathrm{R}_{t+1} = 100 \cdot ((1-TC) \cdot \frac{Close_{t+horizon}}{Close_t} - 1)) \tag{13}$$

- 当前未持仓，且$A_t$为 sell 或 hold 时，奖励为预测期间内的空头收益率：

$$\mathrm{R}_{t+1} = 100 \cdot (1 - \frac{Close_{t+horizon}}{Close_t}) \tag{14}$$

- 当前持多仓，且$A_t$为 sell 时，奖励为预测区间内扣费后的空头收益率：

$$\mathrm{R}_{t+1} = 100 \cdot ((1-TC) \cdot (2 - \frac{Close_{t+horizon}}{Close_t}) - 1) \tag{15}$$

- 当前持多仓，且$A_t$为 buy 或 hold 时，奖励为预测区间内多头收益率：

$$\mathrm{R}_{t+1} = 100 \cdot (\frac{Close_{t+horizon}}{Close_t} - 1) \tag{16}$$

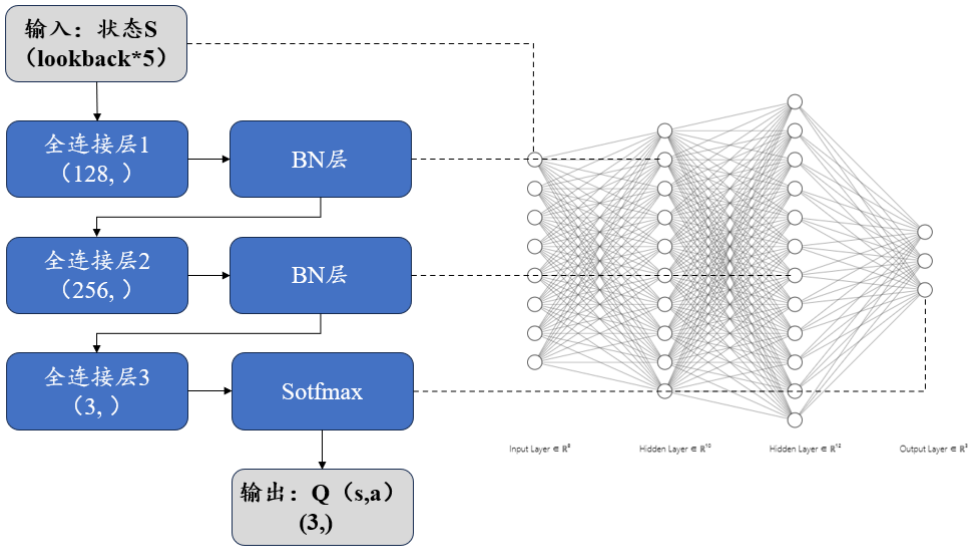
其中 TC 为单边交易费，本项目中取万分之五；$Close_{t+horizon}$为向前 horizon 日收盘价，预测区间 horizon 取 5 个交易日，同时测试 1 和 10。

**折扣因子$\gamma$**

折扣因子$\gamma$取 0.9，同时测试$\gamma = 0.5$和 0.7。

**Q 网络的构建**

Q 网络是 DQN 的核心组件，本项目构建一个 3 层的全链接网络，具体结构如下图所示：



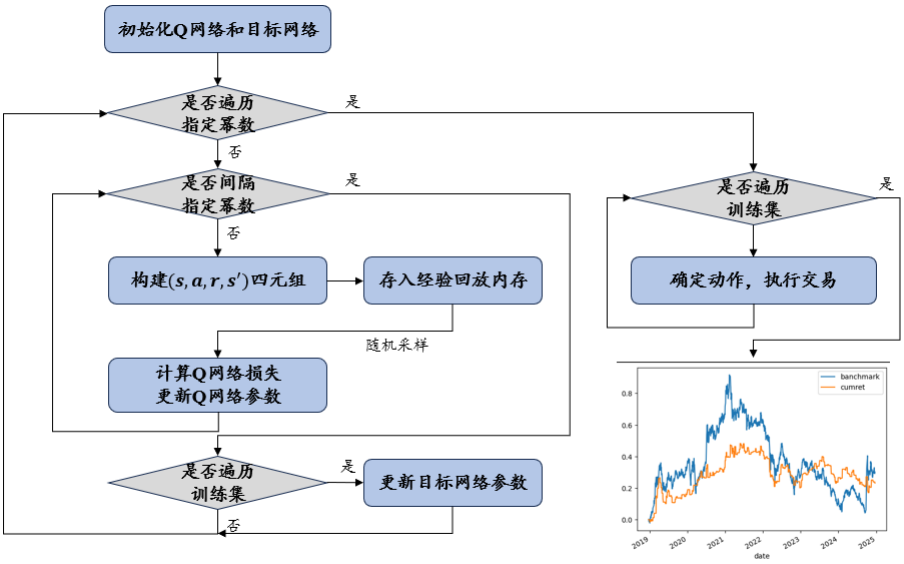

**数据和超参数**

数据集选取 2005 年至 2024 年沪深 300 指数日度行情数据。其中前 70%作为训练集，用来训练智能体；后 30%的年份作为测试集，用于评估智能体表现。

| 类别 | 超参数含义 | 超参数简称 | 取值 |
|---|---|---|---|
| **状态** | 回看区间 | lookback | 5；测试 10,15 |
| **奖励** | 预测区间 | horizon | 5；测试 1,10 |
| | 折扣因子 | $\gamma$ | 0.9；测试 0.5,0.7 |
| **DQN** | 经验回放内存 | replay memory size | 32；测试 16,64 |
| | 训练容量 | batch size | 16 |
| | 迭代次数 | episodes | 30 |
| | 目标网络更新频率 | target network update frequency | 5 |
| **Q 网络** | 优化器 | optimizer | Adam |
| | 学习率 | learning rate | 0.001 |
| | 梯度下降动量 | betas | (0.9, 0.999) |
| | 损失函数 | loss functions | smooth_l1_loss |
| | 梯度范围 | clamp | [-1, 1] |
| **$\epsilon$-贪心算法** | 初值 | $\epsilon$ start | 0.9 |
| | | $\epsilon$ end | 0.05 |
| | | $\epsilon$ decay | 500 |
| **环境** | 交易费用 | TC | 5e-4 |

**策略逻辑：**

项目整体技术路线图如下：




1. **初始化 Q 网格和目标网格：**随机初始化 Q 网络$Q(s,a;\theta)$的参数$\theta$，将$\theta$复制给目标网络$Q(s,a;\theta^*)$的参数$\theta^*$。
2. **数据预处理，获取状态$s$：**每个交易日$t$，计算$t-lookback+1$至$t$日指数开高低收价格和成交量相对过去 252 日数据的 Z 分数，作为该日的状态向量集和 S。
3. 遍**历训练集，构建经验回放元组：**按时间顺序，遍历训练集内每个交易日。通过 Q 网络计算该日状态 s 的动作价值$Q(s,a;\theta)$，通过$\epsilon$-贪心算法得到动作$a$；根据动作和奖励规则，确定奖励$r$，即$t$至$t+horizon$日多头或空头收益：将$t+horizon$日状态视作新的状态$s'$。由此得到每个交易日的$(s,a,r,s')$四元组。
4. **存入回放内存：**将该条经验存入回放内存。当数据内存填满时，删除最早的一条历史数据。
5. **经验回放，优化 Q 网络：**在经验回放中进行随机采样，得到一个 batch 的训练集数据。基于 Q 网络和目标网络计算 Q 网络损失$L(\theta)$，采用优化器更新 Q 网络参数$\theta$。
6. **间隔指定轮次更新神经网络参数：**每完整遍历一轮训练集，视作一轮训练。每隔指定轮数，将训练网络 Q 网络的参数$\theta$复制给$\theta'$。当训练轮数达到指定轮数，停止训练。

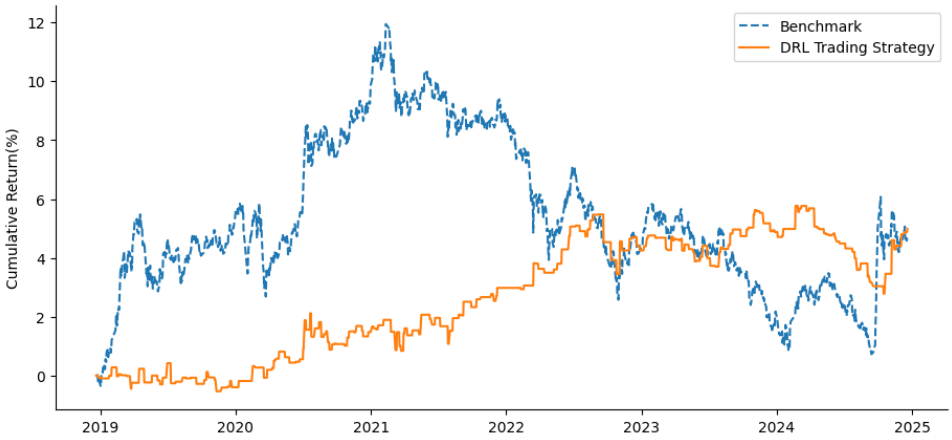
7. **测试集回测：**按时间顺序遍历测试集中的每一个交易日。根据该日状态 S 及训练好的 Q 网络计算动作价值，选择动作价值最高的动作$argmax_a Q(s,a;\theta)$。当处于空仓状态时，若动作为$sell$或$hold$则继续保持空仓，若动作为 buy 则于次日开盘做多。当处于做多状态时，若动作为$buy$或$hold$则继保持做多，若动作为$sell$则于次日开盘平仓。

为控制不同随机数对强化学习结果的影响，本项目采用生成十组随机种子进行并行计算生成一组交易信号，对每日生成的交易信号取出现频率最高的一种作为最后强化学习智能体采取的行动信号。

## 回测结果展示

| | 年化收益率 | 年化波动率 | 夏普比率 | 最大回撤 | Calmar 比率 | 累计收益率 | 信息比率 |
|---|---|---|---|---|---|---|---|
| **原始超参数$\gamma=0.9$，回放内存=64，回看区间=5，预测区间=5** | | | | | | | |
| **$\gamma=0.9$** | 5.34% | 10.61% | 0.50 | 43.07% | 0.05% | 30.74% | -15.92% |
| **折扣因子$\gamma=5$** | | | | | | | |
| **$\gamma=0.5$** | 5.57% | 10.57% | 0.53 | 43.07% | 0.05% | 32.42% | -13.24% |
| **$\gamma=0.7$** | 4.11% | 16.62% | 0.25 | 45.13% | 0.04% | 17.21% | -27.19% |
| **回放内存=64** | | | | | | | |
| **Capacity:32** | -1.45% | 11.64% | -0.12 | 45.00% | -0.01% | -11.59% | -97.83% |
| **Capacity:16** | -1.05% | 11.69% | -0.09 | 45.00% | -0.01% | -9.52% | -92.84% |
| **回看区间=5** | | | | | | | |
| **lookback:10** | 3.44% | 10.51% | 0.33 | 45.37% | 0.03% | 17.76% | -38.61% |
| **lookback:15** | 2.86% | 10.22% | 0.28 | 43.35% | 0.03% | 14.21% | -45.19% |
| **预测区间=5** | | | | | | | |
| **horizion:1** | 5.73% | 11.32% | 0.51 | 44.70% | 0.05% | 32.97% | -15.34% |
| **horizion:10** | 0.23% | 11.14% | 0.02 | 45.60% | 0.00% | -2.24% | -75.66% |

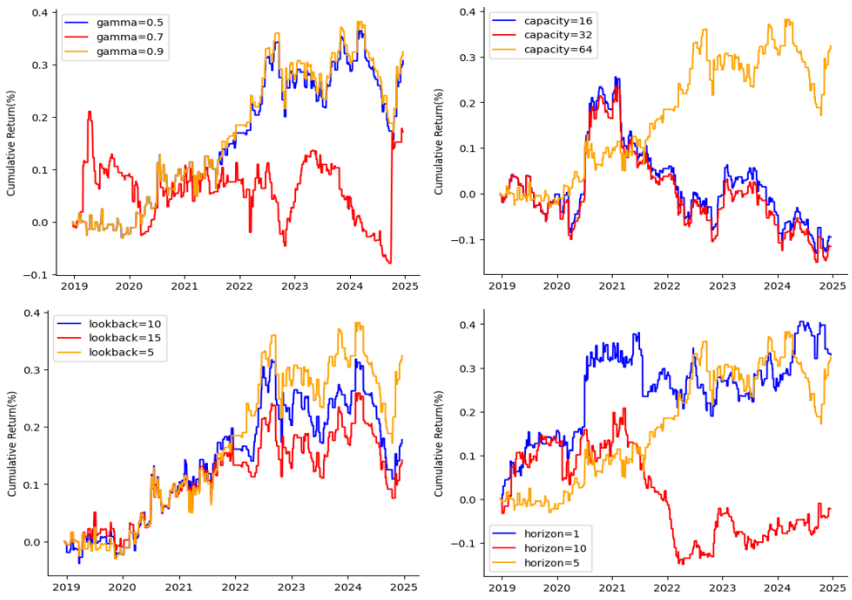
原始超参数样本外回测表现




原始超参数为：折扣因子$\gamma = 0.9$，回放内存$capacity = 32$，回看区间$lookback = 5$，预测区间$horizon = 5$。择时策略的样本外年化收益率 5.34%，夏普比率为 0.5，最大回撤 43%。

**不同超参数对结果的影响**

在这里只对超参数进行了简单的调试和样本外测试。根据结果可以看出强化学习模型本身对参数敏感，参数调整后模型表现也表表现出了不同程度的变化，对比之下可以认为，原始参数的设定已经可以达到一个比较好的结果。




## 总结

本项目使用基于深度强化学习（DQN）的方法构建沪深 300 指数的日频择时策略。有别于传统的机器学习方法，强化学习不存在“标准答案”，而是针对长期收益最大化的目标进行试错学习。其核心思想是通过智能体与环境的交互从反馈的奖励信号中“学习”进而做出正确决策。本文训练 DQN 智能体进行沪深 300 指数的择时策略，在原始超参数下择时策略的样本外年化收益率 5.34%，夏普比率为 0.5，最大回撤 43%。

当前，强化学习方法存在的主要局限为表现稳定性有待进一步提高，通时存在泛化瓶颈。同时发现本文研究发现，华泰金工研报存在将测试集归入训练集美化结果的问题，在我们的研究中，当正确构建经验回放方法和奖励函数的计算之后。样本外收益表现大约稳定在 5%左右，与研报 40%的年化收益相差较大。本文的结果显然更符合 A 股指数择时的认知和预期收益表现，同时这也符合强化学习泛化能力限制的传统认知。

但本文依旧存在以下未尽之处：本研究仅对上证指数进行择时测试，可扩展至更多可交易标的。我们的状态空间仅采用了原始行情数据，其他技术指标和更丰富的因子以及另类数据。任务类型也不仅局限于量化择时，而是可以拓展到更丰富的因子选股、组合优化等方面。另外，从技术的角度，本文的神经网络仅构造了一个简单的全连接网络，在处理时间序列问题上，LSTM 网络或可以带来更好的表现。强化学习存在过拟合风险，需探索过拟合检验方法，且模型对参数较为敏感。由于训练过程复杂耗时，因此对参数的优化依旧有完善空间。进一步的，虽然设置了第二日开盘买入或卖出等具有可行性的交易策略，但依旧可能存在未准确买入卖出的情况，应进一步考虑滑点概率等显示存在的交易问题，这些都是在未来可以进一步优化和改进的地方[1]。

[1] **本项目使用到的数据和完整代码已同步上传到 Github：https://github.com/renzhongxing9/reinfomentlearning-trading 后续若有改进和更新也会在 Github 项目中修改，欢迎访问并对项目中可能存在的问题批评指正。**

## 附录：代码说明

项目文件夹 Reinforcement_learning_trading 下共包含 6 个代码文件。其中前五个文件是强化学习所需的智能体组件，在这里已经全部调试好并在.py 文件中封装，运行项目代码是无需改动。run.ipynb 文件是本项目的调用文件，在该文件中完成对上述代码的调用和全局超参数的修改

├── Agent.py

├── Environment.py

├── Functions.py

├── Memory.py

├── Model.py

└── run.ipynb

下面展示部分本项目中涉及到的强化学习组件的核心代码：

- **经验回放：**

```python
from collections import namedtuple
import random

Transition = namedtuple('Transition', ('state', 'action', 'reward', 'next_state'))
class ReplayMemory:
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.index = 0
    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)
    def __len__(self):
        return len(self.memory)
    def push(self, state, action, reward, next_state):
        # placeholder
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        # 向memory中添加经验回放
        self.memory[self.index] = Transition(state, action, reward, next_state)
        self.index = (self.index + 1) % self.capacity


```

- **Agent：**

```python
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import warnings
warnings.filterwarnings(action="ignore")

from Model import Net
from Memory import ReplayMemory, Transition

env_a_shape = 0
class DQN(object):
    def __init__(self, n_states, n_actions, capacity, batch_size, epsilon_start, epsilon_end, epsilon_decay, gamma, replace_iter):
        self.actions = n_actions
        self.states = n_states
        self.replace_iter = replace_iter
        self.eval_net, self.target_net = Net(n_actions=self.actions, n_states=self.states), Net(n_actions=self.actions, n_states=self.states)
        self.eval_net.train()
        self.target_net.eval()
        self.optimizer = torch.optim.Adam(
        self.eval_net.parameters(), lr=.001, betas=(0.9, 0.999)
        ) # 优化器仅优化eval_net
        self.loss_func = nn.SmoothL1Loss()
        self.loss = 0
        # memory
        self.learn_step_counter = 0
        self.batch_size = batch_size
        self.capacity = capacity
        self.memory_counter = 0
        self.memory = ReplayMemory(self.capacity)
        # gready algthrim
        self.epsilon_start = epsilon_start
        self.epsilon_end = epsilon_end
        self.epsilon_decay = epsilon_decay
        self.gamma = gamma
        self.epsilon = 0
```

- **环境（继承 OpenAI Gym 框架）：**

```python
from Functions import Get_PosRet, Sharp_Ratio

class StockTradingEnv(gym.Env):
    def __init__(
        self, df, buy_cost_pct, sell_cost_pct, tech_indicator_list, seed, horizon=5):
        self.day = 0 # 交易第几天
        self.df = df # 交易数据
        self.buy_cost_pct = buy_cost_pct # 买入手续费
        self.sell_cost_pct = sell_cost_pct # 卖出手续费
        # self.state_space = state_space
        self.tech_indicator_list = tech_indicator_list # 技术指标(factor)名字的list
        self.action_space = spaces.Discrete(3)
        self.data = self.df.loc[self.day, :]
        self.terminal = False # 是否到达终点
        self.initial = True
        self.horizon = horizon
        self.state = self._initiate_state() # 从df中获取第0天的statement数据
        # initialize reward
        self.reward = 0
        # self.trades = 0
        self.signal_list = []
        self.episode = 0
        # memorize all the total balance change
        self.date_memory = []
        self._seed(seed=seed)
```