Assignment：深度强化学习在投资组合分配中的应用

21307035 邓栩瀛

1.问题描述

通过深度强化学习算法，根据投资组合的当前状态，确定最佳的投资组合分配策略。该算法采用DQN模型，关键组成部分包括：

- **智能体**：代表投资组合经历、智能投顾或投资者。
- **动作**：投资组合权重的分配和再平衡。DQN模型提供Q值，Q值将被转化为投资组合权重。
- **奖励函数**：以夏普比率（Sharpe ratio）为核心，结合标准差作为风险评估指标，用于指导代理优化决策。
- **状态**：状态为基于特定时间窗内证券相关矩阵。相关矩阵式适用于投资组合分配的一个状态变量，包含不同证券的关系信息，可用于投资组合分配。
- **环境**：加密货币交易平台。

2.数据来源

案例使用的加密货币数据来自 Kaggle 平台，包含 2018 年期间每日价格波动。数据集中包括比特币Bitcoin、以太坊Ethereum、瑞波币Ripple、莱特币Litecoin和达什币Dash等流动性较高的加密货币，支持算法在多样化的市场环境下进行训练和验证。

3.算法实现核心思路

3.1 加密货币环境

引入一个模拟环境类`CryptoEnvironment`，为加密货币创建了一个工作环境。

```
class CryptoEnvironment:
    def __init__(self, prices = './data/crypto_portfolio.csv', capital = 1e6):
        self.prices = prices
        self.capital = capital
        self.data = self.load_data()
```

该类具有以下主要功能：

- 函数`getState`：该函数返回状态，即基于回溯期的工具相关性矩阵。该函数根据`is_cov_matrix`或`is_raw_time_series`标志返回状态和历史收益或历史原始数据。

```
    def get_state(self, t, lookback, is_cov_matrix = True, is_raw_time_series = False):
        assert lookback <= t
        decision_making_state = self.data.iloc[t-lookback:t]
        decision_making_state = decision_making_state.pct_change().dropna()
        if is_cov_matrix:
            x = decision_making_state.cov()
            return x
        else:
            if is_raw_time_series:
                decision_making_state = self.data.iloc[t-lookback:t]
            return self.preprocess_state(decision_making_state)
```

- 函数`getReward`：给定投资组合权重和回顾周期，该函数返回投资组合的奖赏（夏普比率）。

```
    def get_reward(self, action, action_t, reward_t, alpha = 0.01):
        def local_portfolio(returns, weights):
            weights = np.array(weights)
            rets = returns.mean()
            covs = returns.cov()
            P_ret = np.sum(rets * weights)
            P_vol = np.sqrt(np.dot(weights.T, np.dot(covs, weights)))
            P_sharpe = P_ret / P_vol
            return np.array([P_ret, P_vol, P_sharpe])
        data_period = self.data[action_t:reward_t]
        weights = action
        returns = data_period.pct_change().dropna()
        sharpe = local_portfolio(returns, weights)[-1]
        sharpe = np.array([sharpe] * len(self.data.columns))
        rew = (data_period.values[-1] - data_period.values[0]) / data_period.values[0]
        return np.dot(returns, weights), sharpe
```

定义`portfolio`函数

```
def portfolio(returns, weights):
    weights = np.array(weights)
    rets = returns.mean() * 252
    covs = returns.cov() * 252
    P_ret = np.sum(rets * weights)
    P_vol = np.sqrt(np.dot(weights.T, np.dot(covs, weights)))
    P_sharpe = P_ret / P_vol
    return np.array([P_ret, P_vol, P_sharpe])
```

3.2 训练模型

用DQN算法，开发强化学习投资组合分配策略，步骤如下：

1.用模块`CryptoEnvironment`的帮助函数`getState`获取当前状态。它返回加密货币时间窗内的相关矩阵。

```
s = env.get_state(np.random.randint(window_size+1, data_length-window_size-1),
window_size)
```

2.用`Agent`类的`act`函数，获取给定状态的行动。行动是加密货币投资组合的权重。

```
action = agent.act(s_)
```

`act`函数定义如下

```
    def act(self, state):
        if not self.is_eval and random.random() ⩽ self.epsilon:
            w = np.random.normal(0, 1, size = (self.portfolio_size, ))
            saved_min = None
            if not self.allow_short:
                w += np.abs(np.min(w))
                saved_min = np.abs(np.min(w))
            saved_sum = np.sum(w)
            w /= saved_sum
            return w , saved_min, saved_sum
        pred = self.model.predict(np.expand_dims(state.values, 0))
        return self.nn_pred_to_weights(pred, self.allow_short)
```

3.用模块`CryptoEnvironment`的`getReward`函数，获取给定行动的奖赏。

```
weighted_returns, reward = env.get_reward(action[0], date1, t)
weighted_returns_equal, reward_equal = env.get_reward(
            np.ones(agent.portfolio_size) / agent.portfolio_size, date1, t)
```

4.用函数`getState`获取下个状态。其状态细节再用在Bellman等式，更新Q函数。

5.将状态、下个状态和行动细节存储到`Agent`对象的存储中。接着，`exeReply`函数再使用该存储。

```
        agent.memory4replay.append((s, s_, action, reward, done))
        if len(agent.memory4replay) ⩾ batch_size:
            agent.expReplay(batch_size)
            agent.memory4replay = []
```

`exeReply`函数的定义

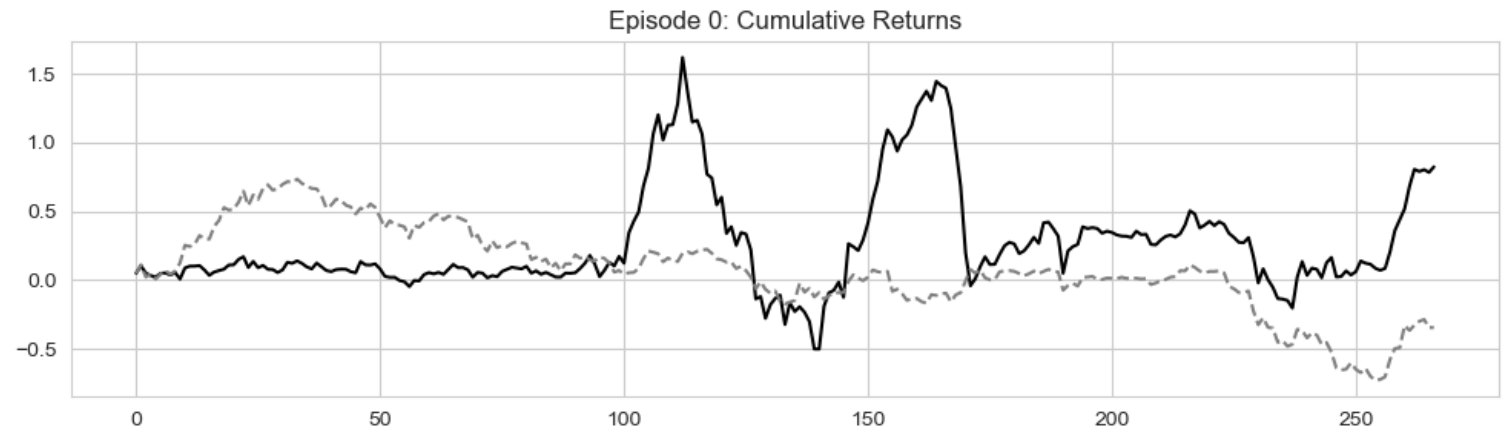
```
    def expReplay(self, batch_size):
        def weights_to_nn_preds_with_reward(action_weights, reward,
    Q_star = np.zeros((self.portfolio_size, self.action_size))):
            Q = np.zeros((self.portfolio_size, self.action_size))
            for i in range(self.portfolio_size):
                if action_weights[i] == 0:
                    Q[i][0] = reward[i] + self.gamma * np.max(Q_star[i][0])
                elif action_weights[i] > 0:
                    Q[i][1] = reward[i] + self.gamma * np.max(Q_star[i][1])
                else:
                    Q[i][2] = reward[i] + self.gamma * np.max(Q_star[i][2])
            return Q
        def restore_Q_from_weights_and_stats(action):
            action_weights, action_min, action_sum = action[0], action[1], action[2]
            action_weights = action_weights * action_sum
            if action_min != None:
                action_weights = action_weights - action_min
            return action_weights
        for (s, s_, action, reward, done) in self.memory4replay:
            action_weights = restore_Q_from_weights_and_stats(action)
            Q_learned_value = weights_to_nn_preds_with_reward(action_weights, reward)
            s, s_ = s.values, s_.values
            if not done:
                Q_star = self.model.predict(np.expand_dims(s_, 0))
                Q_learned_value = weights_to_nn_preds_with_reward(action_weights, reward,
np.squeeze(Q_star))
            Q_learned_value = [xi.reshape(1, -1) for xi in Q_learned_value]
            Q_current_value = self.model.predict(np.expand_dims(s, 0))
            Q = [np.add(a * (1-self.alpha), q * self.alpha) for a, q in
zip(Q_current_value, Q_learned_value)]
            self.model.fit(np.expand_dims(s, 0), Q, epochs=1, verbose=0)

        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

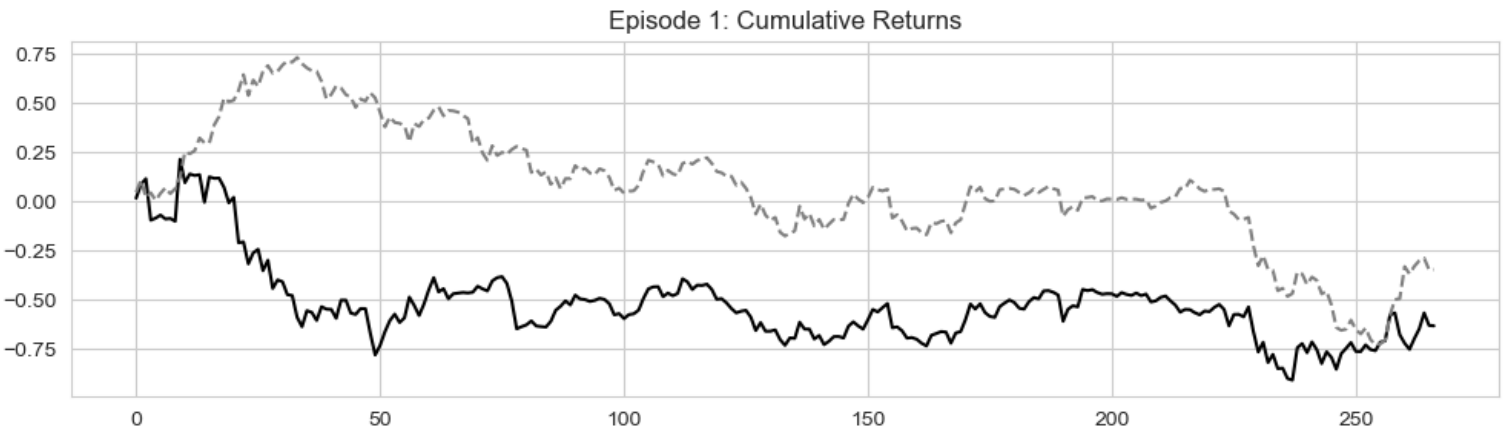
6.检查这批是否完成。每批样例数是由批次样本量变量所定义。如该批未完成，则继续下次迭代；如该批已完成，则调用回放缓存函数，最小化Q预测值和Q目标值的MSE，更新Q函数

3.3 训练结果

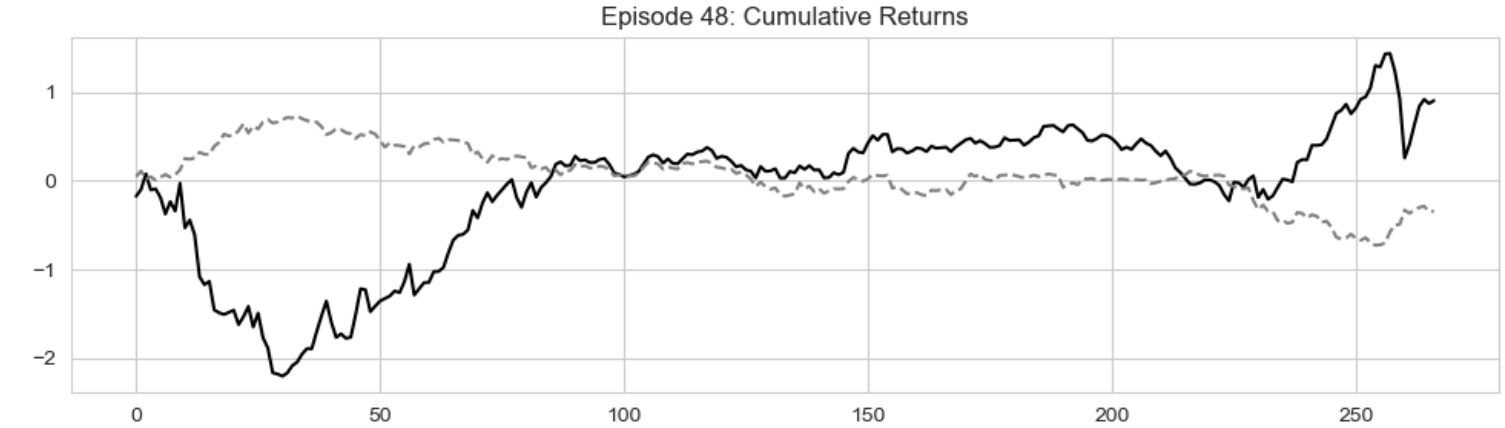
每个episode的训练生成两张图片，第一张图片为一段时间的总累积收益，第二张图片为投资组合中每种加密货币的权重。


Episode 0: Cumulative Returns
1.5
1.0
0.5
0.0
−0.5
0
50
100
150
200
250


Episode 0: Actions
1
0
−1
A
H
B
C
H
S
H
T
K
C
X
T
M
R
P

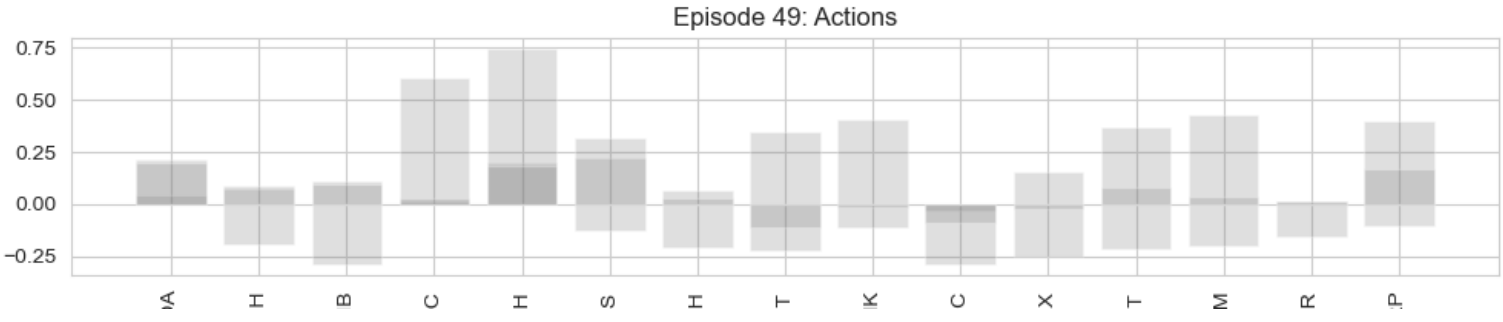

Episode 1: Cumulative Returns
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
50
100
150
200
250


Episode 1: Actions
1.0
0.5
0.0
−0.5
−1.0
A
H
B
C
H
S
H
T
K
C
X
T
M
R
P


Episode 48: Cumulative Returns
1
0
−1
−2
0
50
100
150
200
250


Episode 48: Actions
2
1
0
−1
A
H
B
C
H
S
H
T
K
C
X
T
M
R
P


Episode 49: Cumulative Returns
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
50
100
150
200
250



这些图标描绘首尾各两轮次投资组合的分配结果。黑实线为所得投资组合的表现，而灰虚线为基准组合的表现，以各加密货币权重相同的组合为基准。

在刚开始的第0轮和第1轮，智能体还不清楚其行动的后果，它随机行动并观测结果，结果波动大，第1轮虽趋于稳定，但终不敌基准。由此可知，训练之初每轮累积奖赏波动显著。

对于第48轮和第49轮训练情况，智能体开始从训练中学习，并发现最优策略。总收益相对稳定，其表现超过基准。然而，总投资组合权重仍波动较大，原因是时间序列较短和加密货币资产波动较高。理想情况下，可以增加训练轮次和历史数据长度，来改进训练表现。

3.4 测试结果及结论

模型在最初阶段表现不佳，但整体来看要好于基准，主要是因为在时间窗靠后部分，基准投资组合表现陡降。

（纵坐标为累积收益，横坐标为时间）

查看投资组合和基准组合的收益率、波动率、夏普比率、alpha和beta值

```
EQUAL [-0.0013, 0.0468, -0.5016, -0.0, 1.0]
RL AGENT [-0.0008, 0.0221, -0.5328, -0.0011, -0.2791]
```

总体而言，深度强化学习投资组合各方面都优于基准组合。