第 5 章 现实世界中的强化学习——建立股票/股份交易智能体

基于软件的**深度强化学习**智能体具有巨大的潜力，体现在智能体可以不知疲倦且完美无缺地执行交易策略，而不像人类交易员一样容易受到记忆容量、速度、效率和情感干扰等因素的限制。在股票市场中进行获利的交易需要使用股票代码谨慎地执行买入/卖出交易，在此过程中，要考虑多种市场因素（如交易条件、宏观和微观市场条件等），还要考虑社会、政治和公司的具体变化。在解决现实世界中具有挑战性的问题时，深度强化学习智能体具有很大的潜力，并且存在很多机遇。

然而，由于在现实世界中部署强化学习智能体面临各种挑战，因此，在游戏场景外的现实世界中只有少数几个在游戏之外使用深度强化学习智能体的成功案例。本章的内容主要目的是开发强化学习智能体，用于解决一个有趣且有益的现实问题：股票市场交易。本章提供的内容包含如何实现与 OpenAI Gym 兼容的，具有离散和连续动作空间的自定义股票市场仿真环境。此外，还介绍了如何在股票交易学习环境中构建和训练强化学习智能体。

具体来说，本章将涵盖以下内容：

- 使用真实的证券交易所数据搭建一个股票市场交易强化学习平台；
- 使用价格图表搭建一个股票市场交易强化学习平台；
- 搭建一个高级的股票交易强化学习平台以训练智能体模仿专业交易员。

5.1 技术要求

本书的代码已经在 Ubuntu 18.04 和 Ubuntu 20.04 上进行了广泛的测试，而且可以在安装了 Python 3.6+ 的 Ubuntu 后续版本中正常工作。在安装 Python 3.6 的情况下，搭配每项内容开始时列出的必要 Python 工具包，本书的代码也同样可以在 Windows 和 macOS X 上运行。建议读者创建和使用一个命名为 tf2rl-cookbook 的 Python 虚拟环境来安装工具包以及运行本书的代码。推荐读者安装 Miniconda 或 Anaconda 来管理 Python 虚拟环境。

5.2　使用真实的证券交易所数据搭建一个股票市场交易强化学习平台

股票市场为任何人提供了一个可参与并极具利润潜力的机会。虽然股票市场准入标准低，但并非所有人都能做出持续稳定盈利的交易。主要原因是市场的动态特性以及可能影响人们行为的情感因素，强化学习智能体将情感排除在外，并且可以通过训练来实现持续盈利。本节将实现一个股票市场交易环境，该环境将引导强化学习智能体如何使用真实的股票市场数据进行股票交易。在对它们进行了足够的训练后，就可以部署它们，让它们自动进行交易（和盈利）。

5.2.1　前期准备

为成功运行代码，请确保已经更新到最新版本。需要激活命名为 tf2rl-cookbook 的 Python/Conda 虚拟环境。确保更新的环境与书中代码库中最新的 Conda 环境规范文件（tfrl-cookbook.yml）相匹配。如果以下 import 语句运行没有问题，就可以准备开始了：

```
import os
import random
from typing import Dict

import gym
import numpy as np
import pandas as pd
from gym import spaces

from trading_utils import TradeVisualizer
```

5.2.2　实现步骤

请按照以下步骤实现股票市场交易环境。

（1）初始化环境的可配置参数：

```
env_config = {
  "ticker": "TSLA",
  "opening_account_balance": 1000,
  # Number of steps (days) of data provided to the
  # agent in one observation
  "observation_horizon_sequence_length": 30,
  "order_size": 1,  # Number of shares to buy per
  # buy/sell order
}
```

（2）初始化 StockTradingEnv() 类，并为配置的股票代码加载股票市场数据：

```
class StockTradingEnv(gym.Env):
  def __init__(self, env_config: Dict = env_config):
    """Stock trading environment for RL agents
    Args:
      ticker (str, optional): Ticker symbol for the
      stock. Defaults to "MSFT".
      env_config (Dict): Env configuration values
    """
    super(StockTradingEnv, self).__init__()
    self.ticker = env_config.get("ticker", "MSFT")
    data_dir = os.path.join(os.path.dirname(os.path.\
              realpath(__file__)), "data")
    self.ticker_file_stream = os.path.join(f"{
            data_dir}", f"{self.ticker}.csv")
```

（3）确保股市数据源存在，然后加载数据流：

```
    assert os.path.isfile(
      self.ticker_file_stream
    ), f"Historical stock data file stream not found
     at: data/{self.ticker}.csv"
    # Stock market data stream. An offline file
    # stream is used. Alternatively, a web
    # API can be used to pull live data.
    # Data-Frame: Date Open High Low Close Adj-Close
    # Volume
    self.ohlcv_df = \
      pd.read_csv(self.ticker_file_stream)
```

（4）定义观测和动作空间/环境，以完成初始化函数的定义：

```
    self.opening_account_balance = \
      env_config["opening_account_balance"]
    # Action: 0-> Hold; 1-> Buy; 2 ->Sell;
    self.action_space = spaces.Discrete(3)

    self.observation_features = [
      "Open",
      "High",
      "Low",
      "Close",
      "Adj Close",
      "Volume",
```

```
  ]
  self.horizon = env_config.get(
        "observation_horizon_sequence_length")
  self.observation_space = spaces.Box(
    low=0,
    high=1,
    shape=(len(self.observation_features),
         self.horizon + 1),
    dtype=np.float,
  )
  self.order_size = env_config.get("order_size")
```

（5）实现 get_observation() 函数，以便收集观测：

```
def get_observation(self):
  # Get stock price info data table from input
  # (file/live) stream
  observation = (
    self.ohlcv_df.loc[
      self.current_step : self.current_step + \
        self.horizon,
      self.observation_features,
    ]
    .to_numpy()
    .T
  )
  return observation
```

（6）为了执行交易订单，需要准备好所需的交易内容，所以接下来添加相应的逻辑：

```
def execute_trade_action(self, action):
  if action == 0:  # Hold position
    return
  order_type = "buy" if action == 1 else "sell"

  # Stochastically determine the current stock
  # price based on Market Open & Close
  current_price = random.uniform(
    self.ohlcv_df.loc[self.current_step, "Open"],
    self.ohlcv_df.loc[self.current_step,
              "Close"],
  )
```

（7）初始化完成后，添加买入股票的内容：

```
    if order_type == "buy":
      allowable_shares = \
        int(self.cash_balance / current_price)
      if allowable_shares < self.order_size:
        # Not enough cash to execute a buy order
        # return
      # Simulate a BUY order and execute it at
      # current_price
      num_shares_bought = self.order_size
      current_cost = self.cost_basis * \
               self.num_shares_held
      additional_cost = num_shares_bought * \
                current_price

      self.cash_balance -= additional_cost
      self.cost_basis = (current_cost + \
                 additional_cost) / (
        self.num_shares_held + num_shares_bought
      )
      self.num_shares_held += num_shares_bought

      self.trades.append(
        {
          "type": "buy",
          "step": self.current_step,
          "shares": num_shares_bought,
          "proceeds": additional_cost,
        }
      )
```

（8）同样，添加卖出股票的内容：

```
    elif order_type == "sell":
      # Simulate a SELL order and execute it at
      # current_price
      if self.num_shares_held < self.order_size:
        # Not enough shares to execute a sell
        # order
        return
      num_shares_sold = self.order_size
      self.cash_balance += num_shares_sold * \
                 current_price
```

```
      self.num_shares_held -= num_shares_sold
      sale_proceeds = num_shares_sold * current_price

      self.trades.append(
        {
          "type": "sell",
          "step": self.current_step,
          "shares": num_shares_sold,
          "proceeds": sale_proceeds,
        }
      )
```

（9）更新账户余额：

```
    # Update account value
    self.account_value = self.cash_balance + \
                self.num_shares_held * \
                current_price
```

（10）启动并检查新环境：

```
if __name__ == "__main__":
  env = StockTradingEnv()
  obs = env.reset()
  for _ in range(600):
    action = env.action_space.sample()
    next_obs, reward, done, _ = env.step(action)
    env.render()
```

5.2.3　工作原理

观测值是在 env_config 中指定的某个时间范围内的股票价格信息，包括开盘价、最高价、最低价、收盘价和成交量（OHLCV）。动作空间是离散的，允许执行买入/卖出/持有的交易操作。这是强化学习智能体学习股票市场交易的入门环境。

5.3　使用价格图表搭建一个股票市场交易强化学习平台

人类交易员会查看其价格显视器上的几个指标，以审查和识别潜在的交易。是否可以让智能体也直观地查看价格 K 线图来进行股票交易，而不仅仅是提供表格/CSV 表示？答案是肯定的，本节就介绍如何为强化学习智能体搭建一个具有丰富视觉信息的交易环境。

5.3.1 前期准备

为成功运行代码，请确保已经更新到最新版本。需要激活命名为 tf2rl-cookbook 的 Python/Conda 虚拟环境。确保更新的环境与书中代码库中最新的 Conda 环境规范文件（tfrl-cookbook.yml）相匹配。如果以下 import 语句运行没有问题，就可以准备开始了：

```
import os
import random
from typing import Dict

import cv2
import gym
import numpy as np
import pandas as pd
from gym import spaces

from trading_utils import TradeVisualizer
```

5.3.2 实现步骤

跟随本节内容，即可搭建出一个完整的股票交易强化学习环境，该环境允许智能体处理可视的股票图表并做出交易决策。

（1）配置学习环境如下：

```
env_config = {
  "ticker": "TSLA",
  "opening_account_balance": 100000,
  # Number of steps (days) of data provided to the
  # agent in one observation
  "observation_horizon_sequence_length": 30,
  "order_size": 1,  # Number of shares to buy per
  # buy/sell order
}
```

（2）实现 StockTradingVisualEnv() 类的初始化步骤：

```
class StockTradingVisualEnv(gym.Env):
  def __init__(self, env_config: Dict = env_config):
    """Stock trading environment for RL agents

    Args:
      ticker (str, optional): Ticker symbol for the
      stock. Defaults to "MSFT".
```

```
    env_config (Dict): Env configuration values
  """
  super(StockTradingVisualEnv, self).__init__()
  self.ticker = env_config.get("ticker", "MSFT")
  data_dir = os.path.join(os.path.dirname(os.path.\
            realpath(__file__)), "data")
  self.ticker_file_stream = os.path.join(
        f"{data_dir}", f"{self.ticker}.csv")
  assert os.path.isfile(
    self.ticker_file_stream
  ), f"Historical stock data file stream not found\
    at: data/{self.ticker}.csv"
  # Stock market data stream. An offline file
  # stream is used. Alternatively, a web
  # API can be used to pull live data.
  # Data-Frame: Date Open High Low Close Adj-Close
  # Volume
  self.ohlcv_df = \
    pd.read_csv(self.ticker_file_stream)
```

（3）实现 ___init___() 函数：

```
  self.opening_account_balance = \
    env_config["opening_account_balance"]

  self.action_space = spaces.Discrete(3)

  self.observation_features = [
    "Open",
    "High",
    "Low",
    "Close",
    "Adj Close",
    "Volume",
  ]
  self.obs_width, self.obs_height = 128, 128
  self.horizon = env_config.get(
    "observation_horizon_sequence_length")
  self.observation_space = spaces.Box(
    low=0, high=255, shape=(128, 128, 3),
    dtype=np.uint8,
  )
  self.order_size = env_config.get("order_size")
```

```
    self.viz = None # Visualizer
```

（4）定义环境的 step() 函数：

```
def step(self, action):
  # Execute one step within the trading environment
  self.execute_trade_action(action)
  self.current_step += 1
  reward = self.account_value - \
    self.opening_account_balance  # Profit (loss)
  done = self.account_value <= 0 or \
        self.current_step >= len(
    self.ohlcv_df.loc[:, "Open"].values
  )
  obs = self.get_observation()
  return obs, reward, done, {}
```

（5）实现第（4）步中使用的两个未定义的函数。要实现 get_observation() 函数，需要初始化 TradeVisualizer() 函数。因此，先实现 reset() 函数：

```
def reset(self):
  # Reset the state of the environment to an
  # initial state
  self.cash_balance = self.opening_account_balance
  self.account_value = self.opening_account_balance
  self.num_shares_held = 0
  self.cost_basis = 0
  self.current_step = 0
  self.trades = []
  if self.viz is None:
    self.viz = TradeVisualizer(
      self.ticker,
      self.ticker_file_stream,
      "TFRL-Cookbook Ch4-StockTradingVisualEnv",
    )

  return self.get_observation()
```

（6）实现 get_observation() 函数：

```
def get_observation(self):
  """Return a view of the Ticker price chart as
    image observation
  Returns:
    img_observation (np.ndarray): Image of ticker
```

```
    candle stick plot with volume bars as
    observation
  """
  img_observation = \
    self.viz.render_image_observation(
      self.current_step, self.horizon
    )
  img_observation = cv2.resize(
    img_observation, dsize=(128, 128),
    interpolation=cv2.INTER_CUBIC
  )
  return img_observation
```

（7）在实现智能体所采取的交易动作的执行内容时，可以把交易执行逻辑的实现拆分为接下来的 3 个步骤：

```
def execute_trade_action(self, action):
  if action == 0:  # Hold position
    return
  order_type = "buy" if action == 1 else "sell"

  # Stochastically determine the current stock
  # price based on Market Open & Close
  current_price = random.uniform(
    self.ohlcv_df.loc[self.current_step, "Open"],
    self.ohlcv_df.loc[self.current_step, \
              "Close"],
  )
```

（8）实现执行“buy”订单的内容：

```
  if order_type == "buy":
    allowable_shares = \
      int(self.cash_balance / current_price)
    if allowable_shares < self.order_size:
      return
    num_shares_bought = self.order_size
    current_cost = self.cost_basis * \
            self.num_shares_held
    additional_cost = num_shares_bought * \
              current_price
    self.cash_balance -= additional_cost
    self.cost_basis = (current_cost + \
              additional_cost)/ \
```