Bruce Yang Follow

Oct 26, 2021 · 7 min read · ✦ · Listen

A Data Scientist's Approach for Algorithmic Trading using Deep Reinforcement Learning: An End-to-end Tutorial for Paper Trading

End-to-end tutorial in one Jupyter Notebook: get the data, perform data engineering, train and deploy the DRL agent, place trades, check the portfolio performance

Image by bash__profile on Unsplash

Disclaimer: This is NOT a financial advice, and NOT a recommendation to trade real money. Many broker platforms exist for simulated trading (paper trading) which can be used for building and developing the methods discussed. Please use common sense and always first consult a financial advisor before trading or investing.

This article is prepared by Bruce Yang, Jingyang Rui, and Xiao-Yang Liu.

Paper trading is a MUST for algorithmic trading especially for machine learning trading strategies. Most fund would test and fine-tune their strategies through paper trading before shooting real trades.

In quantitative finance, stock trading is essentially making dynamic decisions, namely to decide *where to trade, at what price, and what quantity* over a highly stochastic and complex stock market. As a result, Deep Reinforcement Learning (DRL) provides useful toolkits for stock trading. Taking many complex financial factors into account, DRL trading agents build a multi-factor model and provide algorithmic trading strategies, which are difficult for human traders.

In this article, we use a single Jupyter notebook to show all the steps for paper trading to test our DRL strategy in real life!

After reading this article you will be able to

- Train a DRL agent on *minute* level data of Dow 30
- Deploy the DRL agent to Alpaca trading API using FinRL
- Place trades on Alpaca using virtual money through Jupyter Notebook

What is Paper Trading?

Paper trading is simulated trading that allows people to 1) practice trading securities; 2) test and fine-tune new investment strategies before apply it in live account; 3) teach beginners how to place orders and learn the trading basics.

However, paper trading does not 1) reflect true emotions in live trading; 2) deal with risk aversion; 3) represent real spreads because it's virtual money not real money.

Part 1: Setup Alpaca Paper Trading API

Image by chris23 on Unsplash

We select Alpaca Trading API mostly because it's free and beginner friendly.

1.1 Key Advantages:

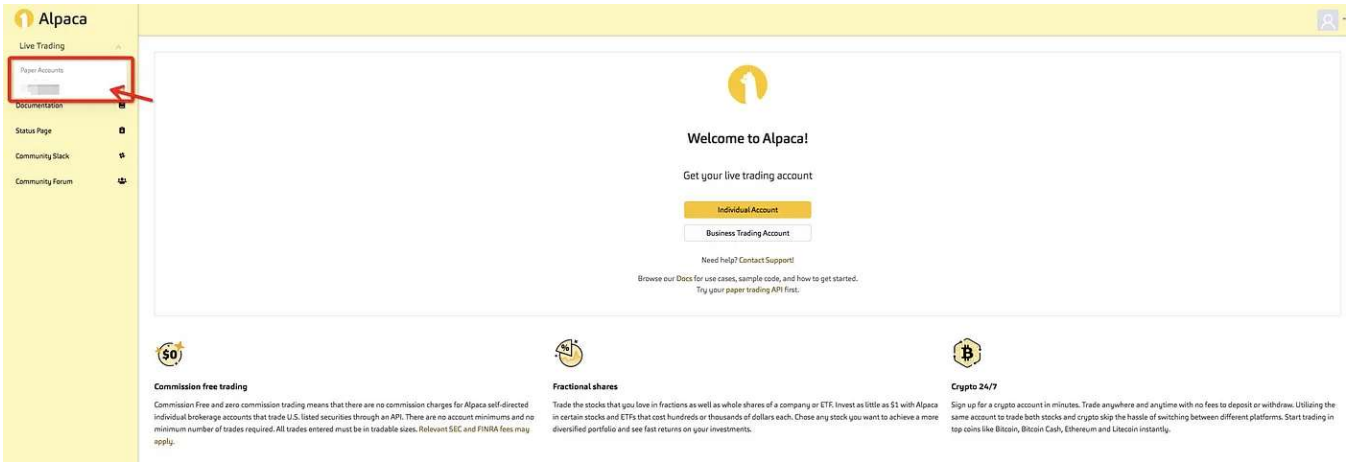
- Commission-free like Robinhood
- Easy to use, good interface
- Good API for algo-trading, beginner friendly
- Unlimited testing of the strategies and bots for free
- No need to be a US resident

> *The free rate limit is 200 requests every minute per API key.*

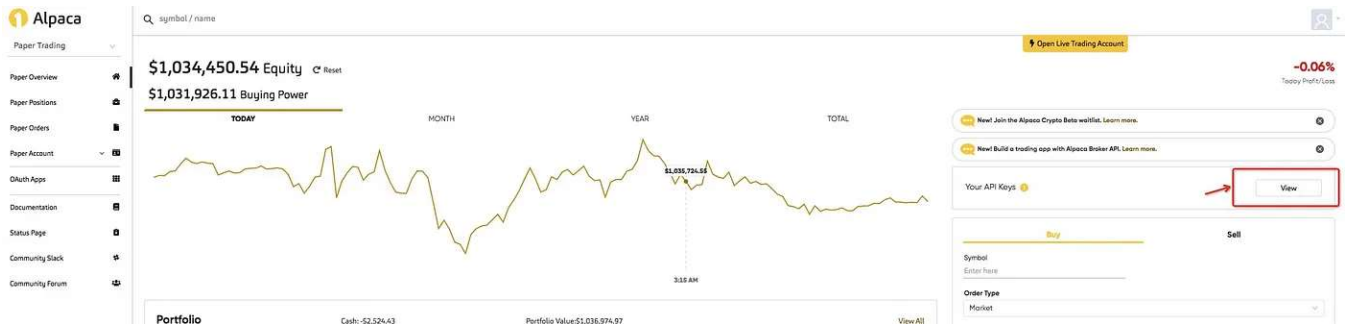
1.2 API Endpoints Setup

In Alpaca's documentation, you'll find all the API calls to submit trades, check the portfolio, and utilizing Python. Here are some good blogs that have very detailed introduction.

To get started in this article, you only need to 1) register an account, 2) generate and keep your **API Key ID** & **Secret ID** (need to regenerate key if forget).



Copyright by AI4Finance-Foundation



Part 2: Get our Deep Reinforcement Learning Agent Ready!

Image by eiskonen on Unsplash

2.0 Install FinRL

FinRL is an **open-source** framework to help practitioners establish the development pipeline of trading strategies based on **deep reinforcement learning (DRL)**. The FinRL framework allows users to plug in and play with **standard DRL algorithms**. Please refer to **this blog** for FinRL installation. This blog uses Google Colab to run Jupyter notebook, please find our code here:

FinRL/FinRL_PaperTrading_Demo.ipynb at master · AI4Finance-Foundation/FinRL

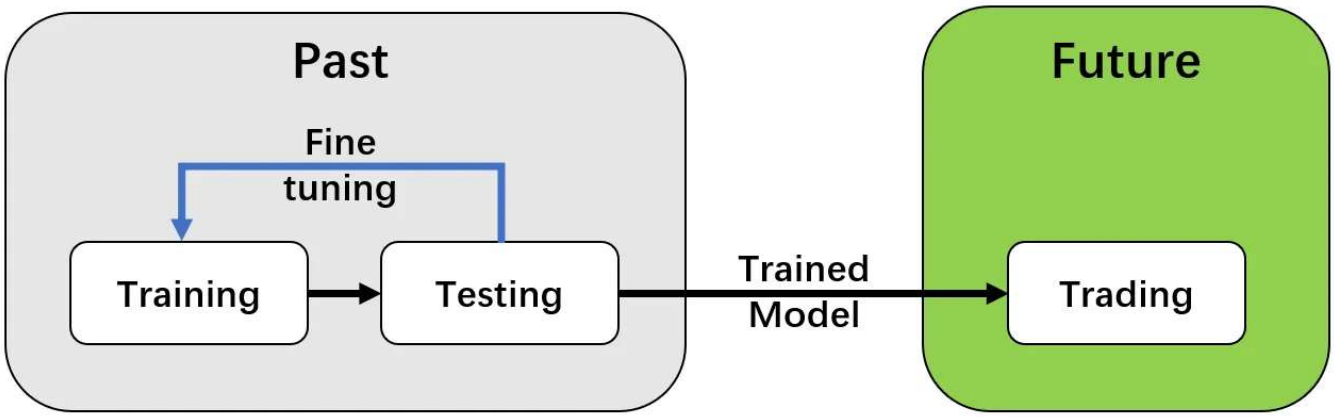
FinRL: Financial Reinforcement Learning. 🔥. Contribute to AI4Finance-Foundation/FinRL development by creating an...

github.com

The "**training-testing**" workflow used by conventional machine learning methods falls short for financial tasks. It splits the data into training set and testing set. On the training data, users select features and tune parameters (k-folds); then make inference on the testing data. However, financial tasks will experience a simulation-to-reality gap between the testing performance and real-live market performance.

Because the testing here is offline backtesting, while the users' goal is to place orders in a real-world market.

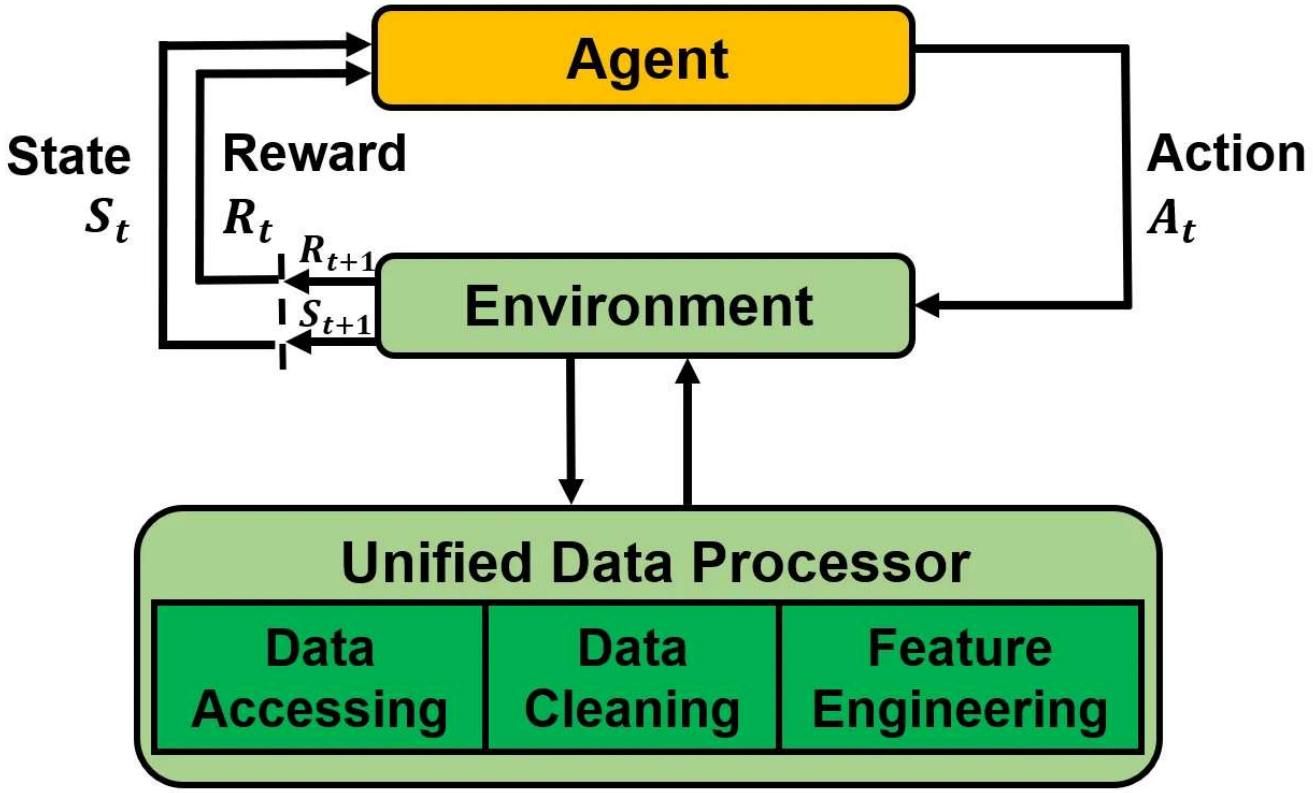
FinRL employs a "**training-testing-trading**" pipeline to reduce the simulation-to-reality gap.



Copyright by AI4Finance-Foundation

We use historical data (time series) for the "**training-testing**" part, which is the same as conventional machine learning tasks, and this **testing** period is for **backtesting** purpose. For the "**trading**" part, we use **live trading APIs**, in this case is Alpaca, allowing users carry out trades directly in a trading system.

2.2 Data Engineering



We establish a standard pipeline for **financial data engineering** in RL, ensuring data of different formats from different sources can be incorporated in a unified framework. Then, we automate this pipeline with a **data processor**, which can *access data, clean data, and extract features* from various data sources with high quality and efficiency. Our data layer provides agility to model deployment.

Step 1: Pick a data source

```
DP = DataProcessor(data_source = 'alpaca',
                   API_KEY = API_KEY,
                   API_SECRET = API_SECRET,
                   APCA_API_BASE_URL = APCA_API_BASE_URL)
```

Step 2: Get the stock ticker list, set start/end date, and specify the time_interval

```
data = DP.download_data(start_date = '2021-10-01',
                        end_date = '2021-10-05',
                        ticker_list = ticker_list,
                        time_interval= '1Min')
```

Step 3: Data Cleaning & Feature Engineering

```
data = DP.clean_data(data)
data = DP.add_technical_indicator(data, TECHNICAL_INDICATORS_LIST)
data = DP.add_vix(data)
```

Step 4: Transform to numpy array format

```
price_array, tech_array, turbulence_array = DP.df_to_array(data,
if_vix='True')
```

```
data.tail(20)
```

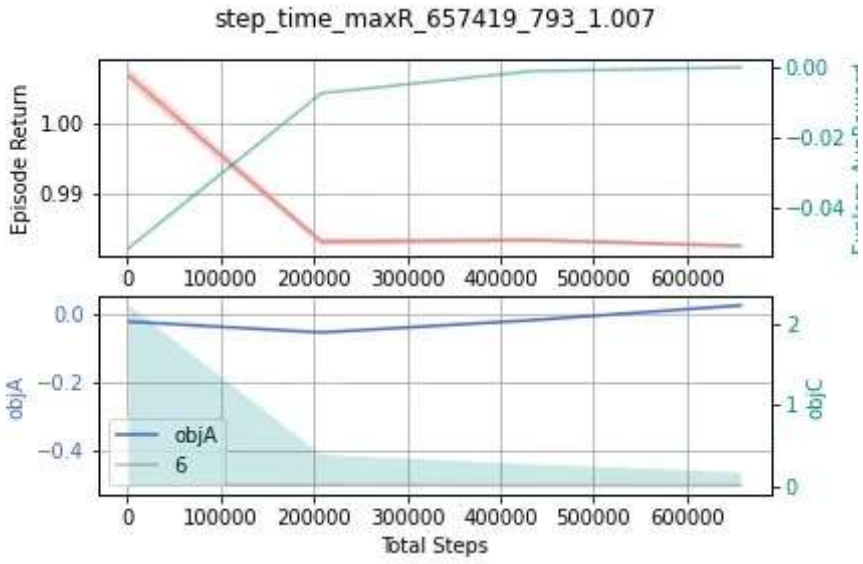
	time	open	high	low	close	volume	tic	macd	boll_ub	boll_lb	rsi_30	cci_30	dx_30	close_30_sma	close_60_sma	VIXY
35080	2021-10-05 15:59:00-04:00	143.140	143.180	143.100	143.160	4180.0	IBM	-0.145148	143.866517	143.059983	30.924818	-160.386451	48.082544	143.534000	143.722417	21.68
35081	2021-10-05 15:59:00-04:00	53.955	53.975	53.935	53.945	6274.0	INTC	-0.038827	54.166900	53.925100	29.721071	-169.470945	57.246117	54.060000	54.096000	21.68
35082	2021-10-05 15:59:00-04:00	159.720	159.730	159.540	159.580	4075.0	JNJ	-0.073306	160.073831	159.657669	34.877315	-306.971514	34.150919	159.888667	160.026250	21.68
35083	2021-10-05 15:59:00-04:00	168.870	168.880	168.650	168.650	7616.0	JPM	-0.140191	169.514417	168.669583	33.354857	-173.244738	42.521277	169.178000	169.313083	21.68
35084	2021-10-05 15:59:00-04:00	53.085	53.090	53.055	53.080	15679.0	KO	-0.012753	53.198559	53.068441	42.715325	-190.357440	29.750107	53.138333	53.137750	21.68
35085	2021-10-05 15:59:00-04:00	245.330	245.330	245.180	245.180	2704.0	MCD	-0.269918	246.739285	245.066715	21.785286	-160.170797	79.728448	246.058333	246.202083	21.68
35086	2021-10-05 15:59:00-04:00	177.960	177.960	177.790	177.790	1754.0	MMM	-0.076775	178.495952	177.863548	34.300943	-232.273583	44.417489	178.213500	178.172583	21.68
35087	2021-10-05 15:59:00-04:00	81.595	81.610	81.575	81.590	8480.0	MRK	-0.054943	81.956850	81.510150	38.831341	-142.857143	25.207846	81.732167	81.815083	21.68
35088	2021-10-05 15:59:00-04:00	288.850	288.880	288.530	288.730	24244.0	MSFT	-0.146467	289.571293	288.706207	39.004141	-188.379205	42.062003	289.225667	289.492667	21.68
35089	2021-10-05 15:59:00-04:00	149.580	149.590	149.465	149.520	6883.0	NKE	-0.111984	150.142632	149.492368	37.775018	-166.219839	60.708595	149.912500	149.907083	21.68
35090	2021-10-05 15:59:00-04:00	42.345	42.355	42.290	42.300	19989.0	PFE	-0.027733	42.429359	42.284641	38.597015	-120.575885	23.939389	42.380000	42.431583	21.68
35091	2021-10-05 15:59:00-04:00	139.430	139.430	139.310	139.310	3316.0	PG	-0.083896	139.923238	139.396262	31.750025	-264.842747	48.807575	139.713667	139.735833	21.68
35092	2021-10-05 15:59:00-04:00	88.780	88.780	88.690	88.740	4767.0	RTX	-0.074596	89.244555	88.704945	34.390595	-186.968291	63.034810	89.004667	89.026250	21.68
35093	2021-10-05 15:59:00-04:00	153.570	153.650	153.570	153.650	1320.0	TRV	-0.038149	153.850881	153.514619	46.886586	-74.290919	24.670390	153.720500	153.740417	21.68
35094	2021-10-05 15:59:00-04:00	393.900	393.900	393.200	393.200	2778.0	UNH	-0.337330	395.924222	393.719278	35.160318	-314.024950	44.275058	394.942833	395.146833	21.68
35095	2021-10-05 15:59:00-04:00	224.340	224.380	224.200	224.270	5364.0	V	-0.232614	225.621581	224.186419	27.798707	-173.469388	71.642315	225.030167	225.164333	21.68
35096	2021-10-05 15:59:00-04:00	54.550	54.555	54.500	54.500	13804.0	VZ	-0.026883	54.660802	54.528698	29.257106	-236.286920	59.868748	54.610167	54.652333	21.68
35097	2021-10-05 15:59:00-04:00	47.100	47.100	47.080	47.090	3300.0	WBA	-0.000467	47.216146	47.085354	47.064782	-62.975239	26.794769	47.125000	47.107500	21.68
35098	2021-10-05 15:59:00-04:00	136.690	136.710	136.600	136.620	5695.0	WMT	-0.076694	137.037700	136.663800	31.557994	-218.758646	61.392805	136.904833	136.971083	21.68
35099	2021-10-05 15:59:00-04:00	61.655	61.670	61.590	61.610	24331.0	XOM	-0.060220	61.912768	61.574732	38.426421	-132.690302	60.013011	61.791167	61.855167	21.68

2.3 Train our DRL agent

We only need to provide some basic parameters and model hyperparameters to the train() function, after training finished, it will output the trained model to the folder we specified (cwd) along with the learning rate plot.

```
#demo for elegantrl
ERL_PARAMS = {"learning_rate": 3e-5,
              "batch_size": 2048,
              "gamma": 0.99,
              "seed":312,
              "net_dimension":512}

train(start_date = '2021-10-11',
      end_date = '2021-10-15',
      ticker_list = ticker_list,
      data_source = 'alpaca',
      time_interval= '1Min',
      technical_indicator_list= TECHNICAL_INDICATORS_LIST,
      drl_lib='elegantrl',
      env=env,
      model_name='ppo',
      API_KEY = API_KEY,
      API_SECRET = API_SECRET,
      APCA_API_BASE_URL = APCA_API_BASE_URL,
      erl_params=ERL_PARAMS,
      cwd='./papertrading_erl',
      total_timesteps=1e5)
```

```
Alpaca successfully connected
Data before 2021-10-11T15:59:00-04:00 is successfully fetched
Data before 2021-10-12T15:59:00-04:00 is successfully fetched
Data before 2021-10-13T15:59:00-04:00 is successfully fetched
Data before 2021-10-14T15:59:00-04:00 is successfully fetched
Data before 2021-10-15T15:59:00-04:00 is successfully fetched
Data clean finished!
Succesfully add technical indicators
Data before 2021-10-11T15:59:00-04:00 is successfully fetched
Data before 2021-10-12T15:59:00-04:00 is successfully fetched
Data before 2021-10-13T15:59:00-04:00 is successfully fetched
Data before 2021-10-14T15:59:00-04:00 is successfully fetched
Data before 2021-10-15T15:59:00-04:00 is successfully fetched
Data clean finished!
Successfully transformed into array
| Remove cwd: ./papertrading_erl
################################################################################
ID     Step    maxR |    avgR    stdR    avgS  stdS |    expR    objC    etc.
0  3.90e+03    1.01 |
0  3.90e+03    1.01 |    1.01     0.0    1948     0 |   -0.05    1.96    0.04  -0.50
| UsedTime:     149 | SavedDir: ./papertrading_erl
```

2.4 Test our DRL agent

The purpose of backtesting is to tune model hyperparameters, make sure that we get a positive and valid return/reward (model converges), and get a good performing strategy.

```
#demo for elegantrl

account_value_erl=
 test(start_date = '2021-10-18',
      end_date = '2021-10-19',
      ticker_list = ticker_list,
```

```
      data_source = 'alpaca',
      time_interval= '1Min',
      technical_indicator_list= TECHNICAL_INDICATORS_LIST,
      drl_lib='elegantrl',
      env=env,
      model_name='ppo',
      API_KEY = API_KEY,
      API_SECRET = API_SECRET,
      APCA_API_BASE_URL = APCA_API_BASE_URL,
      cwd='./papertrading_erl',
      net_dimension = 512)
```

```
Alpaca successfully connected
Data before 2021-10-18T15:59:00-04:00 is successfully fetched
Data before 2021-10-19T15:59:00-04:00 is successfully fetched
Data clean finished!
Succesfully add technical indicators
Data before 2021-10-18T15:59:00-04:00 is successfully fetched
Data before 2021-10-19T15:59:00-04:00 is successfully fetched
Data clean finished!
Successfully transformed into array
price_array:  780
Test Finished!
episode_return 1.0026875870882481
```


2.5 Use full data to train

After the hyperparameters is fixed, we need to use all data available up to the point to train the model because model needs to dynamically adapt to new patterns in data, in case of concept drift.

```
#demo for elegantrl
ERL_PARAMS = {"learning_rate": 3e-5,
              "batch_size": 2048,
              "gamma": 0.99,
              "seed":312,
              "net_dimension":512}

train(start_date = '2021-10-11',
      end_date = '2021-10-19',
      ticker_list = ticker_list,
      data_source = 'alpaca',
      time_interval= '1Min',
      technical_indicator_list= TECHNICAL_INDICATORS_LIST,
      drl_lib='elegantrl',
      env=env,
      model_name='ppo',
      API_KEY = API_KEY,
      API_SECRET = API_SECRET,
```

```
APCA_API_BASE_URL = APCA_API_BASE_URL,
erl_params=ERL_PARAMS,
cwd='./papertrading_erl',
total_timesteps=1e5)
```

Part 3: Deploy our DRL Agent to Alpaca Paper Trading API

3.1 Deploy our agent from the saved file

We load the trained DRL model, and connect it to AlpacaPaperTrading environment to start shooting trades.

```
#demo for elegantrl

paper_trading_erl =
AlpacaPaperTrading(ticker_list = DOW_30_TICKER,
time_interval = '1Min',
drl_lib = 'elegantrl',
agent = 'ppo',
cwd = './papertrading_erl',
net_dim = 512,
state_dim = state_dim,
action_dim= action_dim,
API_KEY = API_KEY,
API_SECRET = API_SECRET,
APCA_API_BASE_URL = APCA_API_BASE_URL,
tech_indicator_list = TECHNICAL_INDICATORS_LIST,
turbulence_thresh=30,
max_stock=1e2)

paper_trading_erl.run()
```

```
Waiting for market to open...
Market opened.
Succesfully add technical indicators
Successfully transformed into array
30
Quantity is 0, order of | 0 AMGN sell | not completed.
Quantity is 0, order of | 0 HON sell | not completed.
Quantity is 0, order of | 0 IBM sell | not completed.
Quantity is 0, order of | 0 PG sell | not completed.
Quantity is 0, order of | 0 TRV sell | not completed.
Quantity is 0, order of | 0 UNH sell | not completed.
Quantity is 0, order of | 0 WBA sell | not completed.
Quantity is 0, order of | 0 WMT sell | not completed.
Quantity is 0, order of | 0 DIS sell | not completed.
Market order of | 20 AXP buy | completed.
Market order of | 33 AAPL buy | completed.
Market order of | 11 CAT buy | completed.
Market order of | 50 CSCO buy | completed.
Market order of | 33 HD buy | completed.
Market order of | 32 INTC buy | completed.
Market order of | 27 JNJ buy | completed.
Market order of | 26 MMM buy | completed.
Market order of | 62 MSFT buy | completed.
Market order of | 27 NKE buy | completed.
Market order of | 44 CRM buy | completed.
Market order of | 28 V buy | completed.
```


3.2 Alpaca Paper Trading Environment for Reinforcement Learning

Environment is the key for reinforcement learning, FinRL provides the Alpaca Paper Trading Env, it connects the DRL agent to the Alpaca Trading API and automatically place trades.

1. Load agent
2. Connect to Alpaca trading API
3. Read trading time interval
4. Read trading settings
5. Initialize account

6. Wait for market to open.

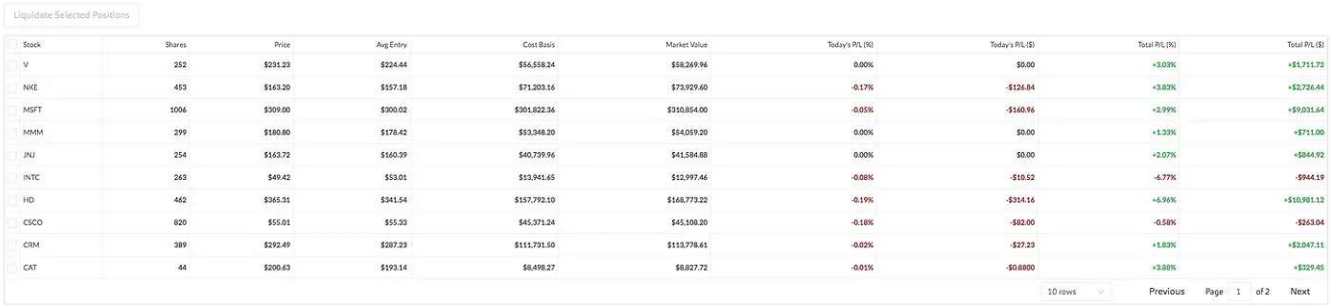
7. Get DRL states.

8. Submit order to place trades.

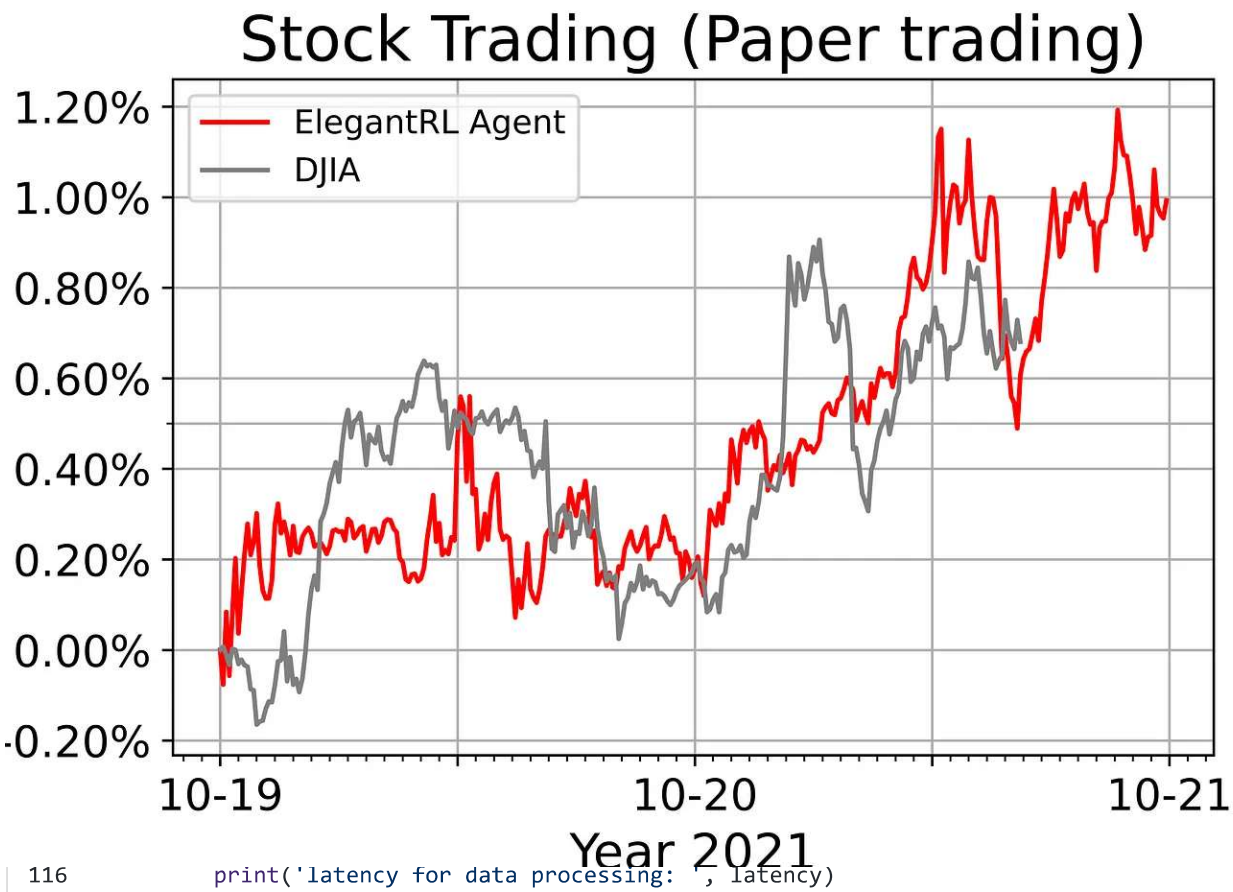
```python
import datetime
import threading
from finrl.neo_finrl.data_processors.processor_alpaca import AlpacaProcessor
from elegantrl.run import *
import alpaca_trade_api as tradeapi
import time
import pandas as pd
import numpy as np
import torch
import sys
import os
import gym

class AlpacaPaperTrading():

    def __init__(self,ticker_list, time_interval, drl_lib, agent, cwd, net_dim,
                 state_dim, action_dim, API_KEY, API_SECRET,
                 APCA_API_BASE_URL, tech_indicator_list, turbulence_thresh=30,
                 max_stock=1e2, latency = None):
        #load agent
        self.drl_lib = drl_lib
        if agent =='ppo':
            if drl_lib == 'elegantrl':
              from elegantrl.agent import AgentPPO
              #load agent
              try:
                  agent = AgentPPO()
                  agent.init(net_dim, state_dim, action_dim)
                  agent.save_or_load_agent(cwd=cwd, if_save=False)
                  self.act = agent.act
                  self.device = agent.device
              except:
                  raise ValueError('Fail to load agent!')

            elif drl_lib == 'rllib':
                from ray.rllib.agents import ppo
                from ray.rllib.agents.ppo.ppo import PPOTrainer

                config = ppo.DEFAULT_CONFIG.copy()
                config['env'] = StockEnvEmpty
                config["log_level"] = "WARN"
                config['env_config'] = {'state_dim':state_dim,
                            'action_dim':action_dim,}
                trainer = PPOTrainer(env=StockEnvEmpty, config=config)
                trainer.restore(cwd)
                try:
                    trainer.restore(cwd)
                    self.agent = trainer
```

```
48                      self.agent = trainer
49                      print("Restoring from checkpoint path", cwd)
50                  except:
51                      raise ValueError('Fail to load agent!')
52
53              elif drl_lib == 'stable_baselines3':
```


```
71              #connect to Alpaca trading API
72              try:
73                  self.alpaca = tradeapi.REST(API_KEY,API_SECRET,APCA_API_BASE_URL, 'v2')
74              except:
```




```
85                  self.time_interval = 60 * 5
86              elif time_interval == '15Min':
87                  self.time_interval = 60 * 15
88              else:
89                  raise ValueError('Time interval input is NOT supported yet.')
90
91              #read trading settings
92              self.tech_indicator_list = tech_indicator_list
93              self.turbulence_thresh = turbulence_thresh
94              self.max_stock = max_stock
95
```




```
116           print('latency for data processing: ', latency)
117           return latency
118
119       def run(self):
120           orders = self.alpaca.list_orders(status="open")
121           for order in orders:
122             self.alpaca.cancel_order(order.id)
123
124           # Wait for market to open.
125           print("Waiting for market to open...")
126           tAMO = threading.Thread(target=self.awaitMarketOpen)
127           tAMO.start()
128           tAMO.join()
129           print("Market opened.")
130           while True:
131
132             # Figure out when the market will close so we can prepare to sell beforehand.
133             clock = self.alpaca.get_clock()
134             closingTime = clock.next_close.replace(tzinfo=datetime.timezone.utc).timestamp()
135             currTime = clock.timestamp.replace(tzinfo=datetime.timezone.utc).timestamp()
136             self.timeToClose = closingTime - currTime
137
138             if(self.timeToClose < (60)):
139               # Close all positions when 1 minutes til market close.
140               print("Market closing soon. Stop trading.")
141               break
142
143               '''# Close all positions when 1 minutes til market close
```

```
            # Close all positions when 1 minutes til market close.
            print("Market closing soon.  Closing positions.")

            positions = self.alpaca.list_positions()
            for position in positions:
              if(position.side == 'long'):
                orderSide = 'sell'
              else:
                orderSide = 'buy'
              qty = abs(int(float(position.qty)))
              respSO = []
              tSubmitOrder = threading.Thread(target=self.submitOrder(qty, position.symbol, or
              tSubmitOrder.start()
              tSubmitOrder.join()

            # Run script again after market close for next trading day.
            print("Sleeping until market close (15 minutes).")
            time.sleep(60 * 15)'''

          else:
            trade = threading.Thread(target=self.trade)
            trade.start()
            trade.join()
            last_equity = float(self.alpaca.get_account().last_equity)
            cur_time = time.time()
            self.equities.append([cur_time,last_equity])
            time.sleep(self.time_interval)

    def awaitMarketOpen(self):
        isOpen = self.alpaca.get_clock().is_open
        while(not isOpen):
          clock = self.alpaca.get_clock()
          openingTime = clock.next_open.replace(tzinfo=datetime.timezone.utc).timestamp()
          currTime = clock.timestamp.replace(tzinfo=datetime.timezone.utc).timestamp()
          timeToOpen = int((openingTime - currTime) / 60)
          print(str(timeToOpen) + " minutes til market open.")
          time.sleep(60)
          isOpen = self.alpaca.get_clock().is_open

    def trade(self):
        state = self.get_state()

        if self.drl_lib == 'elegantrl':
            with torch.no_grad():
                s_tensor = torch.as_tensor((state,), device=self.device)
                a_tensor = self.act(s_tensor)
                action = a_tensor.detach().cpu().numpy()[0]

```

```
            action = (action * self.max_stock).astype(int)

        elif self.drl_lib == 'rllib':
            action = self.agent.compute_single_action(state)

        elif self.drl_lib == 'stable_baselines3':
            action = self.model.predict(state)[0]

        else:
            raise ValueError('The DRL library input is NOT supported yet. Please check your ir

        self.stocks_cd += 1
        if self.turbulence_bool == 0:
            min_action = 10  # stock_cd
            for index in np.where(action < -min_action)[0]:  # sell_index:
                sell_num_shares = min(self.stocks[index], -action[index])
                qty =  abs(int(sell_num_shares))
                respSO = []
                tSubmitOrder = threading.Thread(target=self.submitOrder(qty, self.stockUnivers
                tSubmitOrder.start()
                tSubmitOrder.join()
                self.cash = float(self.alpaca.get_account().cash)
                self.stocks_cd[index] = 0

            for index in np.where(action > min_action)[0]:  # buy_index:
                if self.cash < 0:
                    tmp_cash = 0
                else:
                    tmp_cash = self.cash
                buy_num_shares = min(tmp_cash // self.price[index], abs(int(action[index])))
                qty = abs(int(buy_num_shares))
                respSO = []
                tSubmitOrder = threading.Thread(target=self.submitOrder(qty, self.stockUnivers
                tSubmitOrder.start()
                tSubmitOrder.join()
                self.cash = float(self.alpaca.get_account().cash)
                self.stocks_cd[index] = 0

        else:  # sell all when turbulence
            positions = self.alpaca.list_positions()
            for position in positions:
                if(position.side == 'long'):
                    orderSide = 'sell'
                else:
                    orderSide = 'buy'
                qty = abs(int(float(position.qty)))
                respSO = []
                tSubmitOrder = threading.Thread(target=self.submitOrder(qty, position.symbol,
```

```
                tSubmitOrder.start()
                tSubmitOrder.join()

            self.stocks_cd[:] = 0


    def get_state(self):
        alpaca = AlpacaProcessor(api=self.alpaca)
        price, tech, turbulence = alpaca.fetch_latest_data(ticker_list = self.stockUniverse, t
                                                           tech_indicator_list=self.tech_indicator_l
        turbulence_bool = 1 if turbulence >= self.turbulence_thresh else 0

        turbulence = (self.sigmoid_sign(turbulence, self.turbulence_thresh) * 2 ** -5).astype(

        tech = tech * 2 ** -7
        positions = self.alpaca.list_positions()
        stocks = [0] * len(self.stockUniverse)
        for position in positions:
            ind = self.stockUniverse.index(position.symbol)
            stocks[ind] = ( abs(int(float(position.qty))))

        stocks = np.asarray(stocks, dtype = float)
        cash = float(self.alpaca.get_account().cash)
        self.cash = cash
        self.stocks = stocks
        self.turbulence_bool = turbulence_bool
        self.price = price



        amount = np.array(max(self.cash, 1e4) * (2 ** -12), dtype=np.float32)
        scale = np.array(2 ** -6, dtype=np.float32)
        state = np.hstack((amount,
                    turbulence,
                    self.turbulence_bool,
                    price * scale,
                    self.stocks * scale,
                    self.stocks_cd,
                    tech,
                    )).astype(np.float32)
        print(len(self.stockUniverse))
        return state

    def submitOrder(self, qty, stock, side, resp):
        if(qty > 0):
          try:
            self.alpaca.submit_order(stock, qty, side, "market", "day")
```

```
286                 print("Market order of | " + str(qty) + " " + stock + " " + side + " | completed.'
288               except:
290                 print("Order of | " + str(qty) + " " + stock + " " + side + " | did not go through
291                 resp.append(False)
292             else:
293               print("Quantity is 0, order of | " + str(qty) + " " + stock + " " + side + " | not c
294               resp.append(True)
295
296       @staticmethod
297       def sigmoid_sign(ary, thresh):
298           def sigmoid(x):
299               return 1 / (1 + np.exp(-x * np.e)) - 0.5
300
301           return sigmoid(ary / thresh) * thresh
302
303   class StockEnvEmpty(gym.Env):
304       #Empty Env used for loading rllib agent
305       def __init__(self,config):
306         state_dim = config['state_dim']
307         action_dim = config['action_dim']
308         self.observation_space = gym.spaces.Box(low=-3000, high=3000, shape=(state_dim,), dtype=
309         self.action_space = gym.spaces.Box(low=-1, high=1, shape=(action_dim,), dtype=np.float32
310
311       def reset(self):
312           return
```