FinRL Documentation

Release 0.3.1

FinRL

Mar 03, 2023

CONTENTS

Disclaimer: Nothing herein is financial advice, and NOT a recommendation to trade real money. Please use common sense and always first consult a professional before trading or investing.

AI4Finance community provides this demonstrative and educational resource, in order to efficiently automate trading. FinRL is the first open source framework for financial reinforcement learning.

Reinforcement learning (RL) trains an agent to solve tasks by trial and error, while DRL uses deep neural networks as function approximators. DRL balances exploration (of uncharted territory) and exploitation (of current knowledge), and has been recognized as a competitive edge for automated trading. DRL framework is powerful in solving dynamic decision making problems by learning through interactions with an unknown environment, thus exhibiting two major advantages: portfolio scalability and market model independence. Automated 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. 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.

FinRL provides a framework that supports various markets, SOTA DRL algorithms, benchmarks of many quant finance tasks, live trading, etc.

Join or discuss FinRL with us: AI4Finance mailing list.

Feel free to leave us feedback: report bugs using Github issues or discuss FinRL development in the Slack Channel.

CHAPTER

ONE

INTRODUCTION

Table of Contents

Design Principles

- Plug-and-Play (PnP): modularity; handle different markets (say T0 vs. T+1).
- Completeness and universal: multiple markets; various data sources (APIs, Excel, etc); user-friendly variables.
- Avoid hard-coded parameters.
- Closing the sim-real gap using the "training-testing-trading" pipeline: simulation for training and connecting real-time APIs for testing/trading.
- Efficient data sampling: accelerate the data sampling process is the key to DRL training! From the ElegantRL project. We know that multi-processing is powerful to reduce the training time (scheduling between CPU + GPU).
- Transparency: a virtual env that is invisible to the upper layer.
- Flexibility and extensibility: inheritance might be helpful here.

Contributions

- FinRL is an open source framework for financial reinforcement learning. Trading environments incorporating market frictions are provided.
- Trading tasks accompanied by hands-on tutorials are available in a beginner-friendly and reproducible fashion. Customization is feasible.
- FinRL has good scalability, with fine-tuned state-of-the-art DRL algorithms. Adjusting the implementations to the rapid changing stock market is well supported.
- Typical use cases are selected to establish benchmarks for the quantitative finance community. Standard back-testing and evaluation metrics are also provided for easy and effective performance evaluation.

With FinRL library, the implementation of powerful DRL trading strategies becomes more accessible, efficient and delightful.

CHAPTER

TWO

FIRST GLANCE

To quickly understand what is FinRL and how it works, you can go through the series Stock_NeurIPS2018, including *Stock_NeurIPS2018_Data.ipynb*, *Stock_NeurIPS2018_Train.ipynb*, *Stock_NeurIPS2018_Backtest.ipynb* in our examples directory (https://github.com/AI4Finance-Foundation/FinRL/tree/master/examples)

This is how we use Deep Reinforcement Learning for Stock Trading from scratch.

Tip: Run the code step by step at Google Colab.

The notebook and the following result is based on our paper *Practical deep reinforcement learning approach for stock trading* Xiong, Zhuoran, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. "Practical deep reinforcement learning approach for stock trading." arXiv preprint arXiv:1811.07522 (2018).

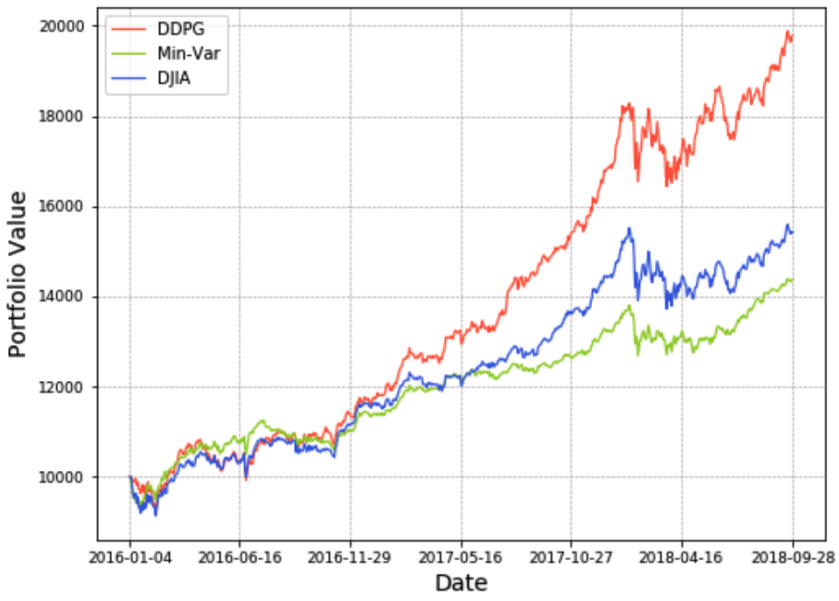


Figure 4: Portfolio value curves of our DDPG scheme, the min-variance portfolio allocation strategy, and the Dow Jones Industrial Average. (Initial portfolio value $\$10,000$).

Table 1: Trading Performance.

	DDPG (ours)	Min-Variance	DJIA
Initial Portfolio Value	$\mathbf{10,000}$	$10,000$	$10,000$
Final Portfolio Value	$\mathbf{19,791}$	$14,369$	$15,428$
Annualized Return	$\mathbf{25.87\%}$	15.93%	16.40%
Annualized Std. Error	$\mathbf{13.62\%}$	9.97%	11.70%
Sharpe Ratio	$\mathbf{1.79}$	1.45	1.27

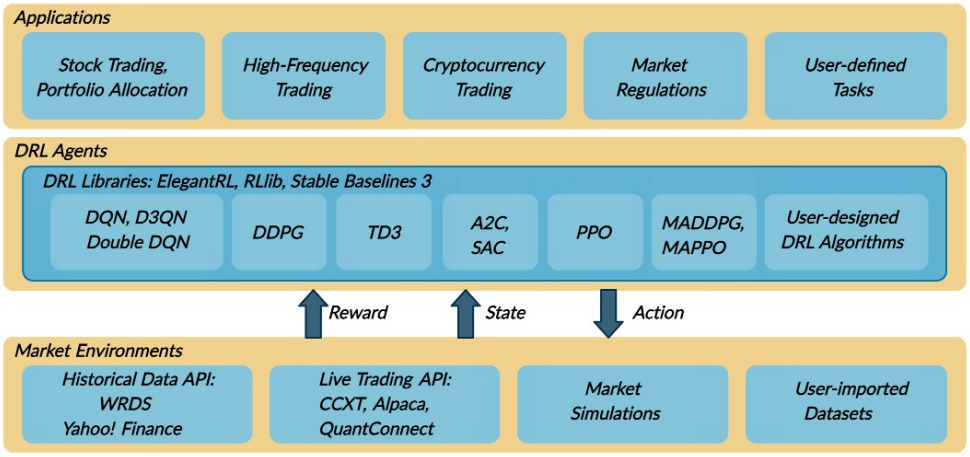
CHAPTER

THREE

THREE-LAYER ARCHITECTURE

After the first glance of how to establish our task on stock trading using DRL, know we are introducing the most central idea of FinRL.

FinRL library consists of three layers: **market environments (FinRL-Meta)**, **DRL agents** and **applications**. The lower layer provides APIs for the upper layer, making the lower layer transparent to the upper layer. The agent layer interacts with the environment layer in an exploration-exploitation manner, whether to repeat prior working-well decisions or to make new actions hoping to get greater cumulative rewards.



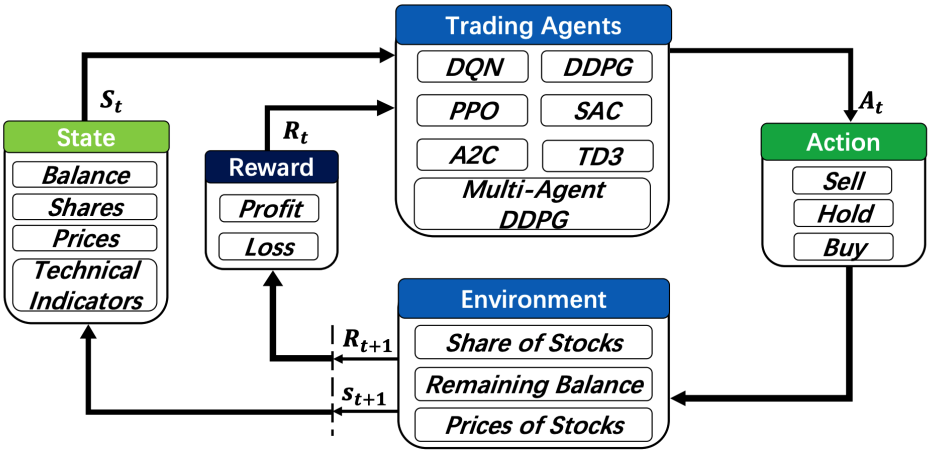
Our construction has following advantages:

Modularity: Each layer includes several modules and each module defines a separate function. One can select certain modules from a layer to implement his/her stock trading task. Furthermore, updating existing modules is possible.

Simplicity, Applicability and Extendibility: Specifically designed for automated stock trading, FinRL presents DRL algorithms as modules. In this way, FinRL is made accessible yet not demanding. FinRL provides three trading tasks as use cases that can be easily reproduced. Each layer includes reserved interfaces that allow users to develop new modules.

Better Market Environment Modeling: We build a trading simulator that replicates live stock markets and provides backtesting support that incorporates important market frictions such as transaction cost, market liquidity and the investor's degree of risk-aversion. All of those are crucial among key determinants of net returns.

A high level view of how FinRL construct the problem in DRL:



Please refer to the following pages for more specific explanation:

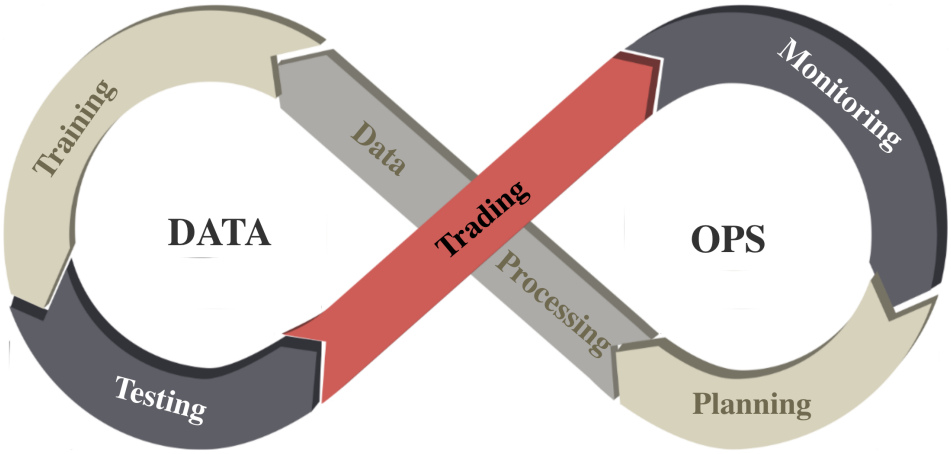
3.1 1. Stock Market Environments

Considering the stochastic and interactive nature of the automated stock trading tasks, a financial task is modeled as a Markov Decision Process (MDP) problem. FinRL-Meta first preprocesses the market data, and then builds stock market environments. The environemnt observes the change of stock price and multiple features, and the agent takes an action and receives the reward from the environment, and finally the agent adjusts its strategy accordingly. By interacting with the environment, the smart agent will derive a trading strategy to maximize the long-term accumulated rewards (also named as Q-value).

Our trading environments, based on OpenAI Gym, simulate the markets with real market data, using time-driven simulation. FinRL library strives to provide trading environments constructed by datasets across many stock exchanges.

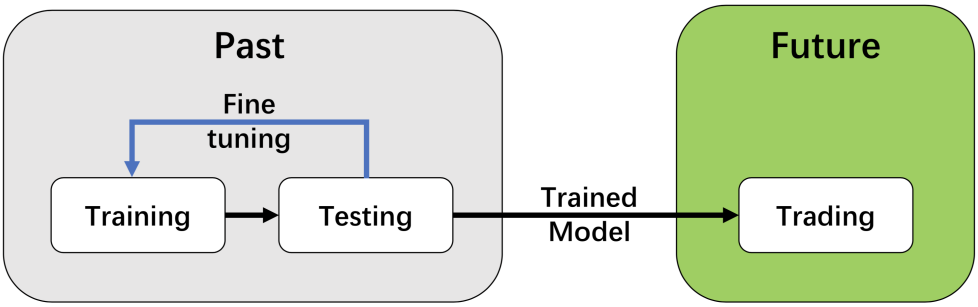
In the Tutorials and Examples section, we will illustrate the detailed MDP formulation with the components of the reinforcement learning environment.

The application of DRL in finance is different from that in other fields, such as playing chess and card games; the latter inherently have clearly defined rules for environments. Various finance markets require different DRL algorithms to get the most appropriate automated trading agent. Realizing that setting up a training environment is time-consuming and laborious work, FinRL provides market environments based on representative listings, including NASDAQ-100, DJIA, S&P 500, SSE 50, CSI 300, and HSI, plus a user-defined environment. Thus, this library frees users from tedious and time-consuming data pre-processing workload. We know that users may want to train trading agents on their own data sets. FinRL library provides convenient support to user-imported data and allows users to adjust the granularity of time steps. We specify the format of the data. According to our data format instructions, users only need to pre-process their data sets.



We follow the DataOps paradigm in the data layer.

- 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**.
- 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.
- We employ a **training-testing-trading pipeline**. The DRL agent first learns from the training environment and is then validated in the validation environment for further adjustment. Then the validated agent is tested in historical datasets. Finally, the tested agent will be deployed in paper trading or live trading markets. First, this pipeline **solves the information leakage problem** because the trading data are never leaked when adjusting agents. Second, a unified pipeline **allows fair comparisons** among different algorithms and strategies.

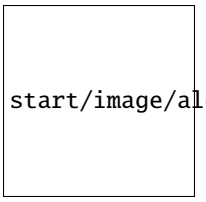


For data processing and building environment for DRL in finance, AI4Finance has maintained another project: FinRL-Meta.

3.2 2. DRL Agents

FinRL contains fine-tuned standard DRL algorithms in ElegantRL, Stable Baseline 3, and RLlib. ElegantRL is a scalable and elastic DRL library that maintained by AI4Finance, with faster and more stable performance than Stable Baseline 3 and RLlib. In the *Three-Layer Architecture* section, there will be detailed explanation about how ElegantRL accomplish its role in FinRL perfectly. If interested, please refer to ElegantRL's GitHub page or documentation.

With those three powerful DRL libraries, FinRL provides the following algorithms for users:



As mentioned in the introduction, FinRL's DRL agents are built by fine-tuned standard DRL algorithms depending on three famous DRL library: ElegantRL, Stable Baseline 3, and RLlib.

The supported algorithms include: DQN, DDPG, Multi-Agent DDPG, PPO, SAC, A2C and TD3. We also allow users to design their own DRL algorithms by adapting these DRL algorithms, e.g., Adaptive DDPG, or employing ensemble methods. The comparison of DRL algorithms is shown in the table bellow:

Algorithms	Input	Output	Type	State-action spaces support	Finance use cases support	Features and Improvements	Advantages
DQN	States	Q-value	Value based	Discrete only	Single stock trading	Target network, experience replay	Simple and easy to use
Double DQN	States	Q-value	Value based	Discrete only	Single stock trading	Use two identical neural network models to learn	Reduce overestimations
Dueling DQN	States	Q-value	Value based	Discrete only	Single stock trading	Add a specialized dueling Q head	Better differentiate actions, improves the learning
DDPG	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Being deep Q-learning for continuous action spaces	Better at handling high-dimensional continuous action spaces
A2C	State action pair	Q-value	Actor-critic based	Discrete and continuous	All use cases	Advantage function, parallel gradients updating	Stable, cost-effective, faster and works better with large batch sizes
PPO	State action pair	Q-value	Actor-critic based	Discrete and continuous	All use cases	Clipped surrogate objective function	Improve stability, less variance, simply to implement
SAC	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Entropy regularization, exploration-exploitation trade-off	Improve stability
TD3	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Clipped double Q-Learning, delayed policy update, target policy smoothing.	Improve DDPG performance
MADDPG	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Handle multi-agent RL problem	Improve stability and performance

Users are able to choose their favorite DRL agents for training. Different DRL agents might have different performance in various tasks.

3.2.1 ElegantRL: DRL library

One sentence summary of reinforcement learning (RL): in RL, an agent learns by continuously interacting with an unknown environment, in a trial-and-error manner, making sequential decisions under uncertainty and achieving a balance between exploration (new territory) and exploitation (using knowledge learned from experiences).

Deep reinforcement learning (DRL) has great potential to solve real-world problems that are challenging to humans, such as gaming, natural language processing (NLP), self-driving cars, and financial trading. Starting from the success

of AlphaGo, various DRL algorithms and applications are emerging in a disruptive manner. The ElegantRL library enables researchers and practitioners to pipeline the disruptive "design, development and deployment" of DRL technology.

The library to be presented is featured with "elegant" in the following aspects:

- Lightweight: core codes have less than 1,000 lines, e.g., helloworld.
- Efficient: the performance is comparable with Ray RLlib.
- Stable: more stable than Stable Baseline 3.

ElegantRL supports state-of-the-art DRL algorithms, including discrete and continuous ones, and provides user-friendly tutorials in Jupyter notebooks. The ElegantRL implements DRL algorithms under the Actor-Critic framework, where an Agent (a.k.a, a DRL algorithm) consists of an Actor network and a Critic network. Due to the completeness and simplicity of code structure, users are able to easily customize their own agents.

Please refer to ElegantRL's GitHub page or documentation for more details.

3.3 3. Applications

CHAPTER

FOUR

INSTALLATION

4.1 MAC OS

4.1.1 Step 1: Install Anaconda

-Download Anaconda Installer, Anaconda has everything you need for Python programming.

-Follow Anaconda's instruction: macOS graphical install, to install the newest version of Anaconda.

-Open your terminal and type: *'which python'*, it should show:

```
/Users/your_user_name/opt/anaconda3/bin/python
```

It means that your Python interpreter path has been pinned to Anaconda's python version. If it shows something like this:

```
/Users/your_user_name/opt/anaconda3/bin/python
```

It means that you still use the default python path, you either fix it and pin it to the anaconda path (try this blog), or you can use Anaconda Navigator to open a terminal manually.

4.1.2 Step 2: Install Homebrew

-Open a terminal and make sure that you have installed Anaconda.

-Install Homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
```

4.1.3 Step 3: Install OpenAI

Installation of system packages on Mac requires Homebrew. With Homebrew installed, run the following in your terminal:

```
brew install cmake openmpi
```

4.1.4 Step 4: Install FinRL

Since we are still actively updating the FinRL repository, please install the unstable development version of FinRL using pip:

```
pip install git+https://github.com/AI4Finance-Foundation/FinRL.git
```

4.1.5 Step 5: Install box2d (if using box2d)

Users can try:

```
brew install swig
pip install box2d-py
pip install box2d
pip install Box2D
```

If it raises errors "AttributeError: module '_Box2D' has no attribute 'RAND_LIMIT_swigconstant' ", users can try:

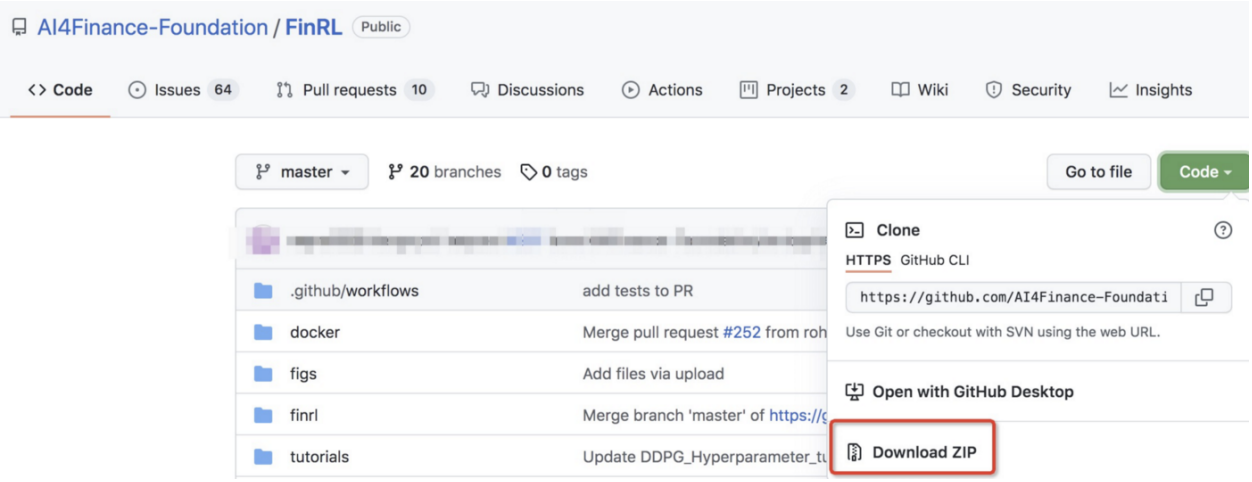
```
pip install box2d box2d-kengz
```

4.1.6 Step 6: Run FinRL

Download the FinRL repository either use terminal:

```
git clone https://github.com/AI4Finance-Foundation/FinRL.git
```

or download it manually



Open Jupyter Notebook through Anaconda Navigator and locate one of the stock trading notebook in FinRL/tutorials you just downloaded. You should be able to run it.

4.2 Ubuntu

4.2.1 Step 1: Install Anaconda

Please follow the steps in this blog

4.2.2 Step 2: Install OpenAI

Open an ubuntu terminal and type:

```
sudo apt-get update && sudo apt-get install cmake libopenmpi-dev python3-dev zlib1g-dev␣
↪libgl1-mesa-glx swig
```

4.2.3 Step 3: Install FinRL

Since we are still actively updating the FinRL repository, please install the unstable development version of FinRL using pip:

```
pip install git+https://github.com/AI4Finance-Foundation/FinRL.git
```

4.2.4 Step 4: Install box2d (if using box2d)

4.2.5 Step 5: Run FinRL

Download the FinRL repository in terminal:

```
git clone https://github.com/AI4Finance-Foundation/FinRL.git
```

Open Jupyter Notebook by typing 'jupyter notebook' in your ubuntu terminal.

Locate one of the stock trading notebook in FinRL/tutorials you just downloaded. You should be able to run it.

4.3 Windows 10

4.3.1 Prepare for install

1. VPN is needed if using YahooFinance in china (pyfolio, elegantRL pip dependencies need pull code, YahooFinance has stopped the service in china). Othewise, please ignore it.
2. python version >=3.7
3. pip remove zipline, if your system has installed zipline, zipline has conflicts with the FinRL.

4.3.2 Step 1: Clone FinRL

```
git clone https://github.com/AI4Finance-Foundation/FinRL.git
```

4.3.3 Step 2: install dependencies

```
cd FinRL
pip install .
```

4.3.4 Step 3: Install box2d (if using box2d)

4.3.5 Step 4: test (If using YahooFinance in China, VPN is needed)

```
python Stock_NeurIPS2018.py
```

4.3.6 Tips for running error

If the following outputs appear, take it easy, since installation is still successful.

1. UserWarning: Module "zipline.assets" not found; multipliers will not be applied to position notionals. Module "zipline.assets" not found; multipliers will not be applied'

If following outputs appear, please ensure that VPN helps to access the YahooFinance

1. Failed download: xxxx: No data found for this date range, the stock may be delisted, or the value is missing.

4.4 Windows 10 (wsl install)

4.4.1 Step 1: Install Ubuntu on Windows 10

Please check this video for detailed steps:

4.4.2 Step 2: Install Anaconda

Please follow the steps in this blog

4.4.3 Step 3: Install OpenAI

Open an ubuntu terminal and type:

```
sudo apt-get update && sudo apt-get install cmake libopenmpi-dev python3-dev zlib1g-dev␣
↪libgl1-mesa-glx
```

4.4.4 Step 4: Install FinRL

Since we are still actively updating the FinRL repository, please install the unstable development version of FinRL using pip:

```
pip install git+https://github.com/AI4Finance-Foundation/FinRL.git
```

4.4.5 Step 5: Install box2d (if using box2d)

4.4.6 Step 6: Run FinRL

Download the FinRL repository in terminal:

```
git clone https://github.com/AI4Finance-Foundation/FinRL.git
```

Open Jupyter Notebook by typing 'jupyter notebook' in your ubuntu terminal. Please see jupyter notebook

Locate one of the stock trading notebook in FinRL/tutorials you just downloaded. You should be able to run it.

CHAPTER

FIVE

QUICK START

Open `main.py`

```python
import os
from typing import List
from argparse import ArgumentParser
from finrl import config
from finrl.config_tickers import DOW_30_TICKER
from finrl.config import (
    DATA_SAVE_DIR,
    TRAINED_MODEL_DIR,
    TENSORBOARD_LOG_DIR,
    RESULTS_DIR,
    INDICATORS,
    TRAIN_START_DATE,
    TRAIN_END_DATE,
    TEST_START_DATE,
    TEST_END_DATE,
    TRADE_START_DATE,
    TRADE_END_DATE,
    ERL_PARAMS,
    RLlib_PARAMS,
    SAC_PARAMS,
    ALPACA_API_KEY,
    ALPACA_API_SECRET,
    ALPACA_API_BASE_URL,
)

# construct environment
from finrl.meta.env_stock_trading.env_stocktrading_np import StockTradingEnv


def build_parser():
    parser = ArgumentParser()
    parser.add_argument(
        "--mode",
        dest="mode",
        help="start mode, train, download_data" " backtest",
        metavar="MODE",
        default="train",
    )
```

(continues on next page)

(continued from previous page)

```
    return parser


# "./" will be added in front of each directory
def check_and_make_directories(directories: List[str]):
    for directory in directories:
        if not os.path.exists("./" + directory):
            os.makedirs("./" + directory)



def main():
    parser = build_parser()
    options = parser.parse_args()
    check_and_make_directories([DATA_SAVE_DIR, TRAINED_MODEL_DIR, TENSORBOARD_LOG_DIR,␣
↪RESULTS_DIR])

    if options.mode == "train":
        from finrl import train

        env = StockTradingEnv

        # demo for elegantrl
        kwargs = {}  # in current meta, with respect yahoofinance, kwargs is {}. For␣
↪other data sources, such as joinquant, kwargs is not empty
        train(
            start_date=TRAIN_START_DATE,
            end_date=TRAIN_END_DATE,
            ticker_list=DOW_30_TICKER,
            data_source="yahoofinance",
            time_interval="1D",
            technical_indicator_list=INDICATORS,
            drl_lib="elegantrl",
            env=env,
            model_name="ppo",
            cwd="./test_ppo",
            erl_params=ERL_PARAMS,
            break_step=1e5,
            kwargs=kwargs,
        )
    elif options.mode == "test":
        from finrl import test
        env = StockTradingEnv

        # demo for elegantrl
        kwargs = {}  # in current meta, with respect yahoofinance, kwargs is {}. For␣
↪other data sources, such as joinquant, kwargs is not empty

        account_value_erl = test(
            start_date=TEST_START_DATE,
            end_date=TEST_END_DATE,
            ticker_list=DOW_30_TICKER,
```

(continues on next page)

(continued from previous page)

```
            data_source="yahoofinance",
            time_interval="1D",
            technical_indicator_list=INDICATORS,
            drl_lib="elegantrl",
            env=env,
            model_name="ppo",
            cwd="./test_ppo",
            net_dimension=512,
            kwargs=kwargs,
        )
    elif options.mode == "trade":
        from finrl import trade
        env = StockTradingEnv
        kwargs = {}
        trade(
            start_date=TRADE_START_DATE,
            end_date=TRADE_END_DATE,
            ticker_list=DOW_30_TICKER,
            data_source="yahoofinance",
            time_interval="1D",
            technical_indicator_list=INDICATORS,
            drl_lib="elegantrl",
            env=env,
            model_name="ppo",
            API_KEY=ALPACA_API_KEY,
            API_SECRET=ALPACA_API_SECRET,
            API_BASE_URL=ALPACA_API_BASE_URL,
            trade_mode='backtesting',
            if_vix=True,
            kwargs=kwargs,
        )
    else:
        raise ValueError("Wrong mode.")


## Users can input the following command in terminal
# python main.py --mode=train
# python main.py --mode=test
# python main.py --mode=trade
if __name__ == "__main__":
    main()
```

Run the library:

```
python main.py --mode=train # if train. Use DOW_30_TICKER by default.
python main.py --mode=test  # if test. Use DOW_30_TICKER by default.
python main.py --mode=trade # if trade. Users should input your alpaca parameters in␣
↪config.py
```

Choices for `--mode`: start mode, train, download_data, backtest

CHAPTER

SIX

BACKGROUND

6.1 Why FinRL-Meta?

Finance is a particularly difficult playground for deep reinforcement learning (DRL). Some existing works already showed great potential of DRL in financial applications. However, establishing high-quality market environments and benchmarks on financial reinforcement learning are challenging and highly demanded. Thus, we proposed and started FinRL-Meta.

6.2 Envrionments and Benchmarks

MuJoCo and OpenAI's XLand are famous libraries in the RL area, they built environments for deep reinforcement learning in robotics, games, and common tasks that are widely used in RL academia and industry. However, they barely provide any high quality environments for financial tasks. FinRL-Meta, previously called Neo-FinRL (near real market environments for data driven financial RL), are working to provide hundreds of market environments and tens of benchmarks for financial reinforcement learning.

6.3 Metaverse for financial RL

Achieving the goal of hundreds of market environments and benchmarks discribed above, we are aiming to build a metaverse for financial reinforcement learning. Like XLand, we would provide an open-ended market world with different tasks e.g. stock, cryptocurrency, etc. for agents to explore and learn.

6.4 Contribute to finance

We believe in the potential of deep reinforcement learning. And we hope that after we build the metaverse for financial reinforcement learning, our agents have chance to be a market simulator, or to explore risk assessment or market fragility.

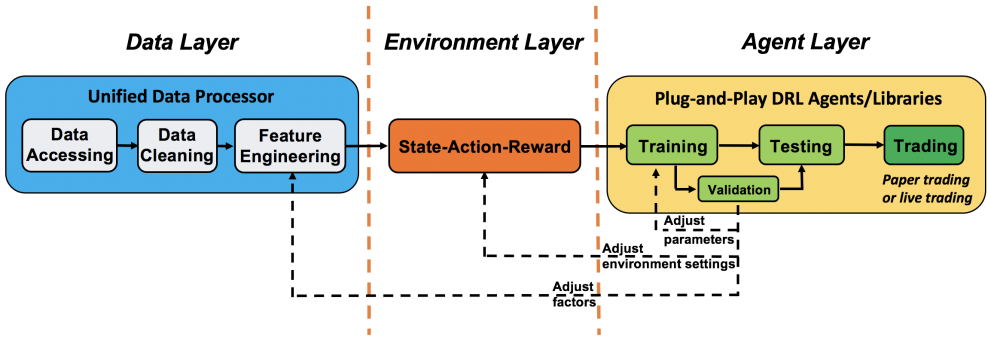
CHAPTER

SEVEN

OVERVIEW

Following the *de facto* standard of OpenAI Gym, we build a universe of market environments for data-driven financial reinforcement learning, namely, FinRL-Meta. We keep the following design principles.

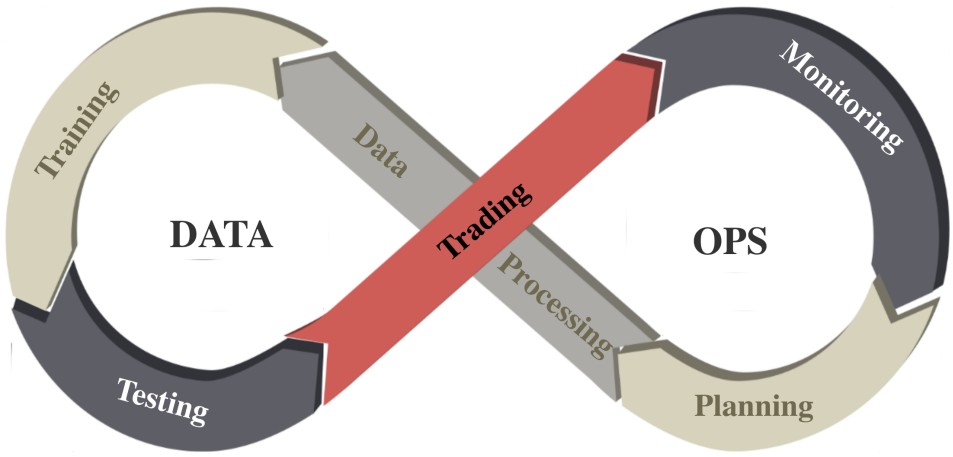
7.1 1. Layered structure



We adopt a layered structure for RL in finance, which consists of three layers: data layer, environment layer, and agent layer. Each layer executes its functions and is relatively independent. There are two main advantages:

1. Transparency: layers interact through end-to-end interfaces to implement the complete workflow of algorithm trading, achieving high extensibility.
2. Modularity: Following the APIs between layers, users can easily customize their own functions to substitute default functions in any layer.

7.2 2. DataOps Paradigm

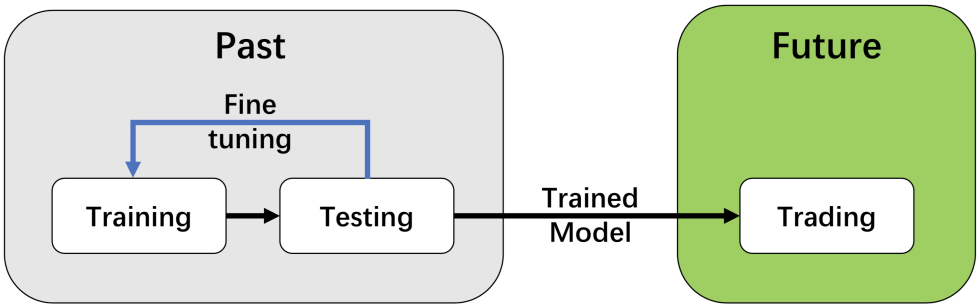


DataOps paradigm is a set of practices, processes and technologies that combined: automated data engineering & agile development. It helps reduce the cycle time of data engineering and improves data quality. To deal with financial big data, we follow the DataOps paradigm and implement an automatic pipeline:

1. Task planning, such as stock trading, portfolio allocation, cryptocurrency trading, etc
2. Data processing, including data accessing and cleaning, and feature engineering.
3. Training-testing-trading, where DRL agent takes part in.
4. Performance monitoring, compare the performance of DRL agent with some baseline trading strategies.

With this pipeline, we are able to continuously produce dynamic market datasets.

7.3 3. Training-testing-trading pipeline:



We employ a training-testing-trading pipeline that the DRL approach follows a standard end-to-end pipeline. The DRL agent is first trained in a training dataset and fined-tuned (adjusting hyperparameters) in a testing dataset. Then, backtest the agent (on historical dataset), or deploy in a paper/live trading market.

This pipeline address the information leakage problem by separating the training/testing-trading periods the agent never see the data in backtesting or paper/live trading stage.

And such a unified pipeline allows fair comparison among different algorithms.

7.4 4. Plug-and-play

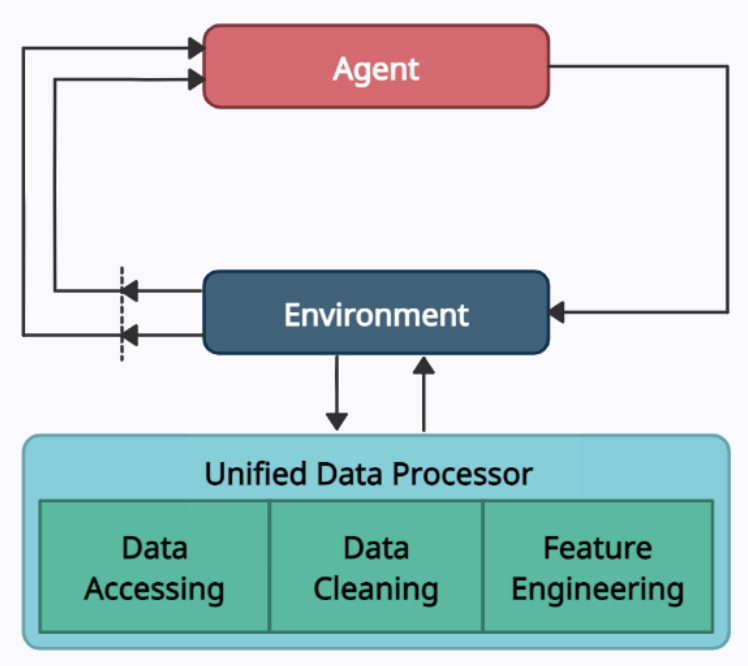
In the development pipeline, we separate market environments from the data layer and the agent layer. Any DRL agent can be directly plugged into our environments, then will be trained and tested. Different agents can run on the same benchmark environment for fair comparisons. Several popular DRL libraries are supported, including ElegantRL, RLlib, and SB3.

CHAPTER

EIGHT

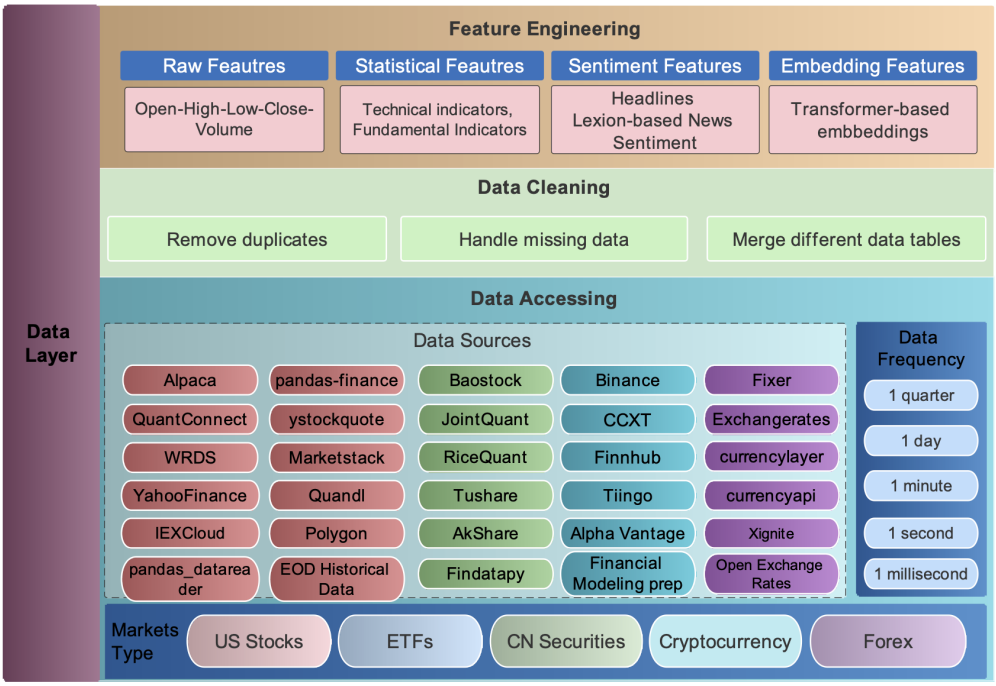
DATA LAYER

In the data layer, we use a unified data processor to access data, clean data, and extract features.



8.1 Data Accessing

We connect data APIs of different platforms and unify them using a FinRL-Meta data processor. Users can access data from various sources given the start date, end date, stock list, time interval, and kwargs.



8.2 Data Cleaning

Raw data retrieved from different data sources are usually of various formats and have erroneous or NaN data (missing data) to different extents, making data cleaning highly time-consuming. In FinRL-Meta, we automate the data cleaning process.

The cleaning processes of NaN data are usually different for various time frequencies. For Low-frequency data, except few stocks with extremely low liquidity, the few NaN values usually mean suspension during that time interval. While for high-frequency data, NaN values are pervasive, which usually means no transaction during that time interval. To reduce the simulation-to-reality gap considering of data efficiency, we provide different solutions for these two cases.

In the low-frequency case, we directly delete the rows with NaN values, reflecting suspension in simulated trading environments. However, it is not suitable to directly delete rows with NaN values in high-frequency cases.

In our test of downloading 1-min OHLCV data of DJIA 30 companies from Alpaca during 2021–01–01~2021–05–31, there were 39736 rows for the raw data. However, after dropping rows with NaN values, only 3361 rows are left.

The low data efficiency of the dropping method is unacceptable. Instead, we take an improved forward filling method. We fill the open, high, low, close columns with the last valid value of close price and the volume column with 0, which is a standard method in practice.

Although this filling method sacrifices the authenticity of the simulated environments, it is acceptable compared to significantly improved data efficiency, especially under tickers with high liquidity. Moreover, this filling method can be further improved using bid, ask prices to reduce the simulation-to-reality gap.

8.3 Feature Engineering

Feature engineering is the last part of the data layer. We automate the calculation of technical indicators by connecting the Stockstats or TAlib library in our data processor. Common technical indicators including Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Average Directional Index (ADX), and Commodity Channel Index (CCI), and so on, are supported. Users can also quickly add indicators from other libraries, or add the user-defined features directly.

Users can add their features by two ways: 1) Write user-defined feature extraction functions directly. The returned features will be added to a feature array. 2) Store the features in a file, and move it to a specified folder. Then, these features will be obtained by reading from the specified file.

CHAPTER

NINE

ENVIRONMENT LAYER

FinRL-Meta follows the OpenAI gym-style to create market environments using the cleaned data from the data layer. It provides hundreds of environments with a common interface. Users can build their environments based on FinRL-Meta environments easily, share their results and compare the strategies' performance. We will add more environments for convenience in the future.

9.1 Incorporating trading constraints to model market frictions

To better simulate real-world markets, we incorporate common market frictions (e.g., transaction costs and investor risk aversion) and portfolio restrictions (e.g., non-negative balance).

- **Flexible account settings**: Users can choose whether to allow buying on margin or short-selling.
- **Transaction cost**: We incorporate the transaction cost to reflect market friction, e.g., 0.1% of each buy or sell trade.
- **Risk-control for market crash**: In FinRL, a financial turbulence index is used to control risk during market crash situations. However, calculating the turbulence index is time-consuming. It may take minutes, which is not suitable for paper trading and live trading. We replace the financial turbulence index with the volatility index (VIX) that can be accessed immediately.

9.2 Multiprocessing training via vector environment

We utilize GPUs for multiprocessing training, namely, the vector environment technique of Isaac Gym, which significantly accelerates the training process. In each CUDA core, a trading agent interacts with a market environment to produce transitions in the form of {state, action, reward, next state}. Then, all the transitions are stored in a replay buffer to update a learner. By adopting this technique, we successfully achieve the multiprocessing simulation of hundreds of market environments to improve the performance of DRL trading agents on large datasets.

CHAPTER

TEN

BENCHMARK

10.1 Performance Metrics

FinRL-Meta provides the following unified metrics to measure the trading performance:

- **Cumulative return:** $R = \frac{V - V_0}{V_0}$, where V is final portfolio value, and V_0 is original capital.
- **Annualized return:** $r = (1 + R)^{\frac{365}{t}} - 1$, where t is the number of trading days.
- **Annualized volatility:** $\sigma_a = \sqrt{\frac{\sum_{i=1}^{n} (r_i - \bar{r})^2}{n-1}}$, where r_i is the annualized return in year i, $\bar{r}$ is the average annualized return, and n is the number of years.
- **Sharpe ratio:** $S = \frac{r - r_f}{\sigma_a}$, where r_f is the risk-free rate.
- **Max. drawdown** The maximal percentage loss in portfolio value.

The following baseline trading strategies are provided for comparisons:

- **Passive trading strategy**, a well-known long-term strategy. The investors just buy and hold selected stocks or indexes without further activities.
- ****Mean-variance and min-variance strategy**, both strategies look for a balance between risks and profits. It selects a diversified portfolio to achieve higher profits at lower risk.
- **Equally weighted strategy**, a portfolio allocation strategy that gives equal weights to different assets, avoiding allocating overly high weights on particular stocks.

10.2 Tutorials in Jupyter Notebooks

For educational purposes, we provide Jupyter notebooks as tutorials to help newcomers get familiar with the whole pipeline. Notebooks can be found here

- Stock trading: We apply popular DRL algorithms to trade multiple stocks.
- Portfolio allocation: We use DRL agents to optimize asset allocation in a set of stocks.
- Cryptocurrency trading: We reproduce the experiment on 10 popular cryptocurrencies.
- Multi-agent RL for liquidation strategy analysis: We reproduce the experiment in [7]. The multi-agent optimizes the shortfalls in the liquidation task, which is to sell given shares of one stock sequentially within a given period, considering the costs arising from the market impact and the risk aversion.
- Ensemble strategy for stock trading: We reproduce the experiment in that employed an ensemble strategy of several DRL algorithms on the stock trading task.

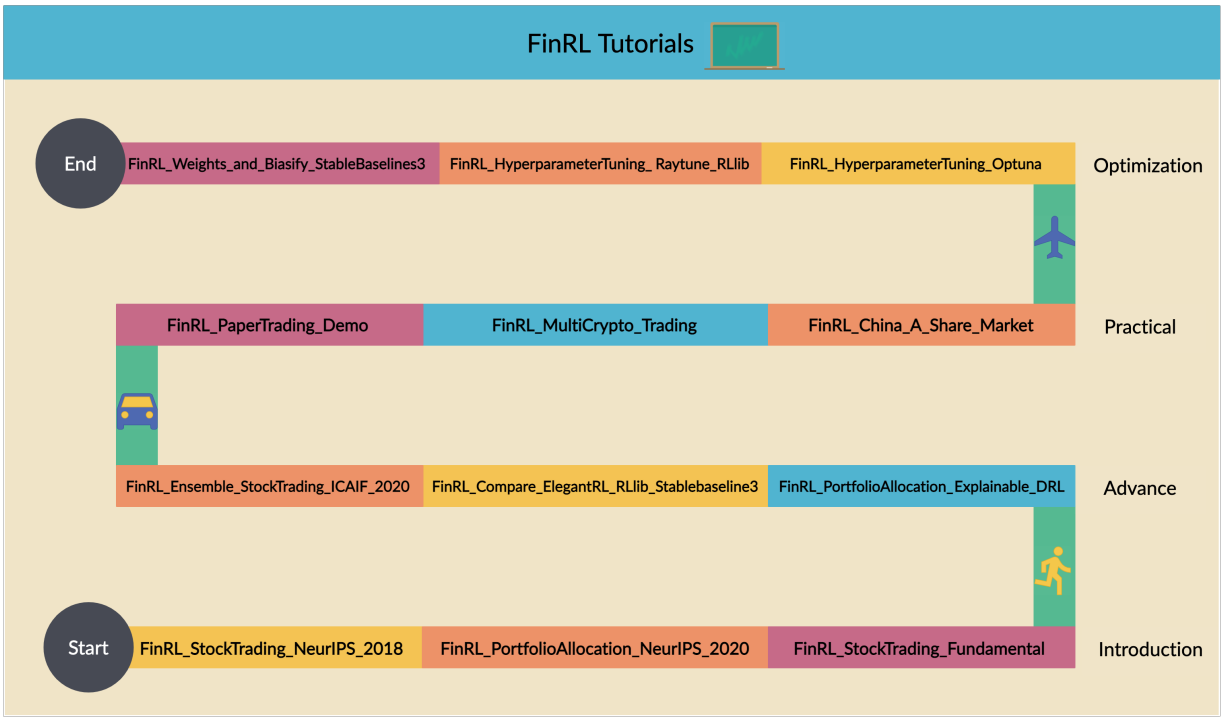
- Paper trading demo: We provide a demo for paper trading. Users could combine their own strategies or trained agents in paper trading.
- China A-share demo: We provide a demo based on the China A-share market data.
- Hyperparameter tuning: We provide several demos for hyperparameter tuning using Optuna or Ray Tune, since hyperparameter tuning is critical for better performance.

CHAPTER

ELEVEN

TUTORIALS GUIDE

Welcome to FinRL's tutorial! In this section, you can walk through the tutorial notebooks we prepared. If you are new to FinRL, we would suggest you the following sequence:



Mission: provide user-friendly demos in notebook or python.

Outline

1-Introduction: basic demos for beginners.

2-Advance: advanced demos, e.g., ensemble stock trading.

3-Practical: paper trading and live trading.

4-Optimization: hyperparameter tuning.

5-Others: other demos.

11.1 1-Introduction

This section is recommend for new comers of FinRL. Users could better learn FinRL in the meantime of running these notebooks.

1. Stock_NeurIPS2018.ipynb,

This is the notebook we recommend new users run first. It goes through a full process of DRL for stock trading using FinRL.

2. China_A_share_market_tushare.ipynb

This notebook demonstrate using FinRL to connect Tushare, using its data of China A share market.

3. FinRL_PortfolioAllocation_NeurIPS_2020.ipynb

This notebook demonstrate using FinRL to do portfolio allocation.

11.2 2-Advance

This section is recommended for users with some familiarity of FinRL or FinRL-Meta (or already run the notebooks in "1-Introduction").

Notebooks in this section includes:

1. FinRL_Compare_ElegantRL_RLlib_Stablebaseline3.ipynb

In this notebook, we compare the three DRL libraries that supported in FinRL. Users who know these DRL libraries might find this interesting.

2. FinRL_Ensemble_StockTrading_ICAIF_2020.ipynb

In this notebook, we implement an "ensemble agent", which is a ensemble of several popular DRL algorithms. Then we compare the performance of the ensemble agent and other DRL agents on the portfolio allocation task.

FinRL_PortfolioAllocation_Explainable_DRL.ipynb.

11.3 3-Practical

This section is recommended for users with some familiarity of FinRL or FinRL-Meta (or already run the notebooks in "1-Introduction"). User could use the provided code for specific task, or design their own task based on existing code.

Notebooks in this section includes:

FinRL_MultiCrypto_Trading.ipynb,

In this notebook, we provide a demo of multiple cryptocurrency trading. It shows a whole process of how to use APIs in FinRL and FinRL-Meta to do cryptocurrency trading.

FinRL_PaperTrading_Demo.ipynb.

In this notebook, we provide a demo of paper trading. It shows a whole process of using FinRL and FinRL-Meta to connect to Alpaca to do paper trading. Note: User need to have their own Alpaca account to run this notebook.

11.4 4-Optimization

This section provideds examples of hyperperameter tuning and connecting cloud platform.

Notebooks in this section includes:

FinRL_HyperparameterTuning_Optuna.ipynb,

FinRL_HyperparameterTuning_Raytune_RLlib.ipynb,

FinRL_HyperparameterTuning_using_Optuna_basic.ipynb,

FinRL_Weights_and_Biasify_StableBaselines3.ipynb.

11.5 5-Others

This section collects uncategorized notebooks such as those made by community members or for some specific usage.

Notebooks in this section includes:

FinRL_demo_docker.ipynb,

tutorial_env_multistock_cashpenalty.ipynb,

tutorial_multistock_docker.ipynb,

tutorial_multistock_variant_2.ipynb,

tutorial_this_works_1_18.ipynb.

CHAPTER

TWELVE

FILE ARCHITECTURE

FinRL's file architecture strictly follow the *Three-layer Architecture*.

```
FinRL
├── finrl (the main folder)
│   ├── applications
│   │       ├── cryptocurrency_trading
│   │       ├── high_frequency_trading
│   │       ├── portfolio_allocation
│   │       └── stock_trading
│   ├── agents
│   │       ├── elegantrl
│   │       ├── rllib
│   │       └── stablebaseline3
│   ├── meta
│   │       ├── data_processors
│   │       ├── env_cryptocurrency_trading
│   │       ├── env_portfolio_allocation
│   │       ├── env_stock_trading
│   │       ├── preprocessor
│   │       ├── data_processor.py
│   │       └── finrl_meta_config.py
│   ├── config.py
│   ├── config_tickers.py
│   ├── main.py
│   ├── train.py
│   ├── test.py
│   ├── trade.py
└──────└── plot.py
```

CHAPTER

THIRTEEN

DEVELOPMENT GUIDE

Git is a commonly used tool in software engineering. PyCharm is a popular IDE for Python, and developers can also choose other IDEs as they like. Now, we take PyCharm as an example. This setup with PyCharm makes it easy to work on all of AI4Finance-Foundation's repositories simultaneously, while allowing easy debugging, committing to the respective repo and creating PRs/MRs.

13.1 Step 1: Download Software

-Download and install Anaconda.

-Download and install PyCharm. The Community Edition (free version) offers everything you need except running Jupyter notebooks. The Full-fledged Professional Edition offers everything. A workaround to run existing notebooks in the Community edition is to copy all notebook cells into .py files. For notebook support, you can consider PyCharm Professional Edition.

-On GitHub, fork FinRL to your private Github repo.

-On GitHub, fork ElegantRL to your private Github repo.

-On GitHub, fork FinRL-Meta to your private Github repo.

-All next steps happen on your local computer.

13.2 Step 2: Git Clone

```
mkdir ~/ai4finance
cd ~/ai4finance
git clone https://github.com/[your_github_username]/FinRL.git
git clone https://github.com/[your_github_username]/ElegantRL.git
git clone https://github.com/[your_github_username]/FinRL-Meta.git
```

13.3 Step 3: Create a Conda Environment

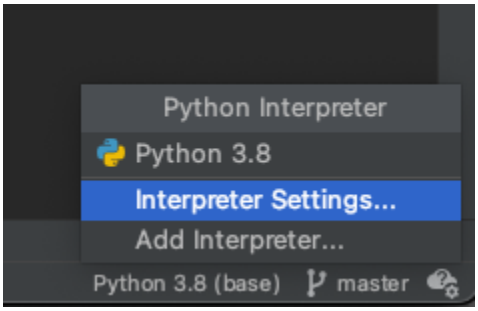
```
cd ~/ai4finance
conda create --name ai4finance python=3.8
conda activate ai4finance

cd FinRL
pip install -r requirements.txt
```

Install ElegantRL using requirements.txt, or open ElegantRL/setup.py in a text editor and pip install anything you can find: gym, matplotlib, numpy, pybullet, torch, opencv-python, and box2d-py.

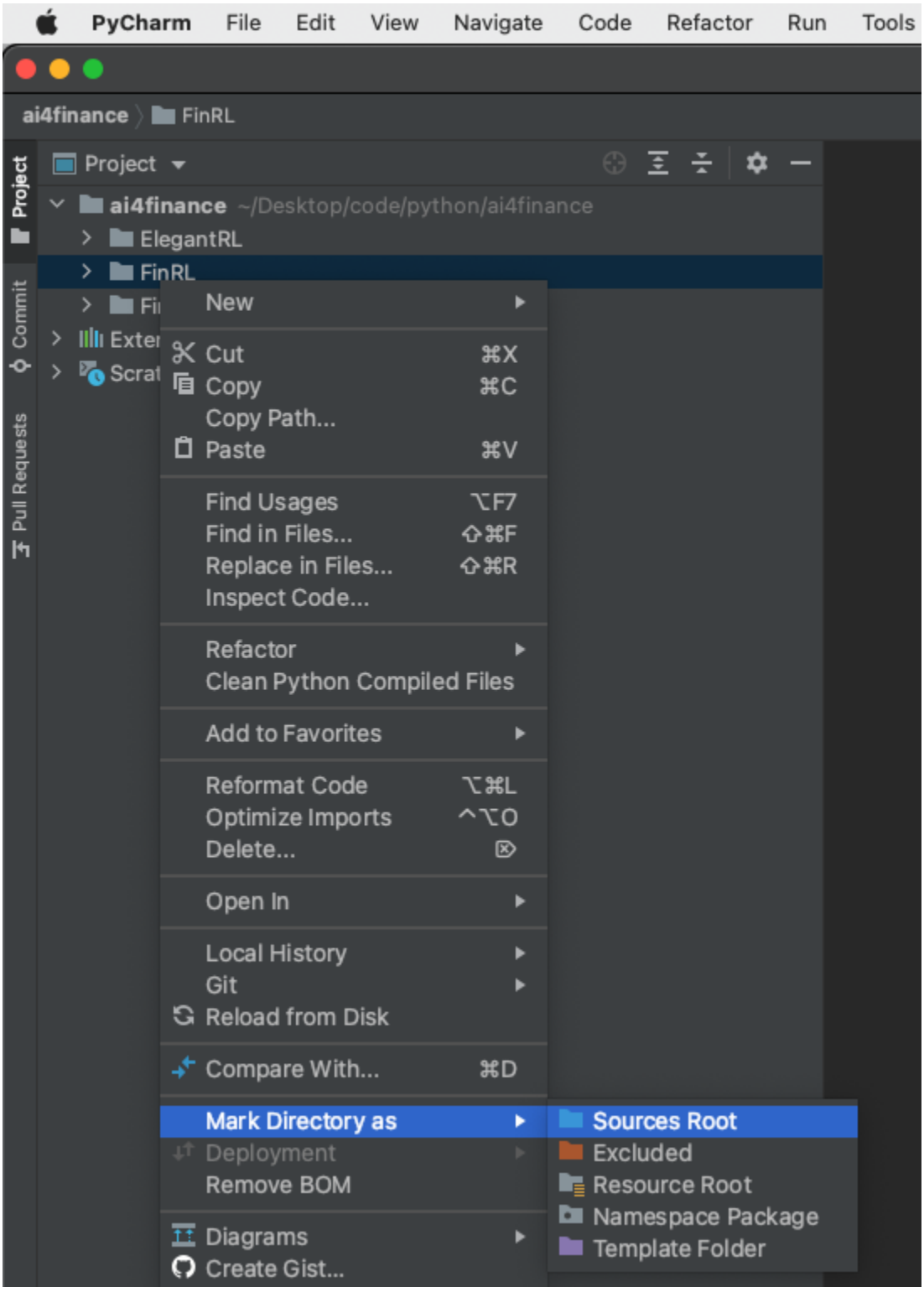
13.4 Step 4: Configure a PyCharm Project

-Launch PyCharm

-File > Open > [ai4finance project folder]



-At the bottom right of the status bar, change or add the interpreter to the ai4finance conda environment. Make sure when you click the "terminal" bar at the bottom left, it shows ai4finance.



-At the left of the screen, in the project file tree:

- Right-click on the FinRL folder > Mark Directory as > Sources Root
- Right-click on the ElegantRL folder > Mark Directory as > Sources Root
- Right-click on the FinRL-Meta folder > Mark Directory as > Sources Root

-Once you run a .py file, you will notice that you may still have some missing packages. In that case, simply pip install them.

For example, we revise FinRL.

```
cd ~/ai4finance
cd ./FinRL
git checkout -b branch_xxx
```

where branch_xxx is a new branch name. In this branch, we revise config.py.

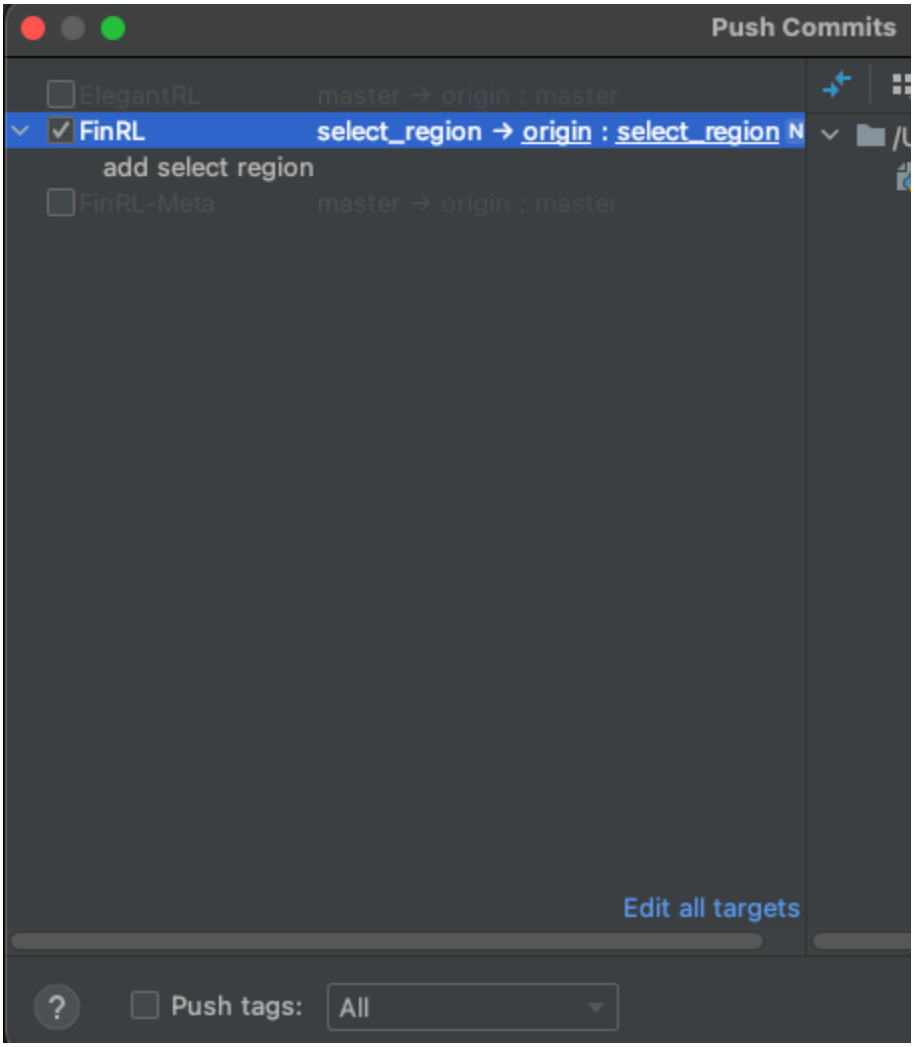
13.5 Step 5: New a Branch

Please new a new branch based on branch "staging" (NOT "master"), which is for all developers. DO NOT directly push codes to the branch "staging" or "master".

13.6 Step 6: Creating Commits and PRs/MRs

-Create commits as you usually do through PyCharm.

-Make sure that each commit covers only 1 of the 3 repo's. Don't create a commit that spans more than one repo, e.g., FinRL and ElegantRL.



-When you do a Git Push, PyCharm will ask you to which of the 3 repos you want to push. Just like the above figure, we select the repo "FinRL".

With respect to creating a pull request (PR) or merge quest (MR), please refer to Create a PR or Opensource Create a PR.

13.7 Step 7: Submit PRs/MRs

When submiting PRs/MRs, please choose the branch "staging", NOT "master".

13.8 Step 8: Merge “staging” to “master”

This step is for managers. If the branch “staging” is stable and works successfully after a series of tests, the managers of this repo will merge it to the branch “master” every 2-4 weeks. To avoid any risk, we hope managers download the “master” branch locally before merging.

CHAPTER

FOURTEEN

CONTRIBUTING GUIDELINES

This project aims to bring a reinforcement learning environment to the trading community. There are always competing priorities among the community, and we want to make sure that we are able to achieve together a project that is reliable, sustainable, and maintainable.

14.1 Guiding Principles

- **We should have reliable codes in this project**
  - reliable code with tests
  - reliable code that works
  - reliable code runs without consuming excessive resources
- We should help each other to achieve SOTA results together
- **We should write clear codes**
  - Code should not be redundant
  - Code should have documentation inline (standard pep format)
  - Code should be organized into classes and functions
- We should leverage outside tools as it makes sense
- We work together, and are kind, patient, and clear in our communication. Jerks are not welcome.

If you see something, say something! * Filing an [issue](https://guides.github.com/features/issues/) is a great way to help improve the project

14.2 Accepting PRs

- You found a bug and a way to fix it
- You have contributed to an issue that was prioritized by the coordinators of this project
- You have new functionality that you're adding that you've written issues for and has documentation + Tests

14.3 PR Guidelines

- Please tag @bruceyang, @spencerromo, or @xiaoyang in every PR. (P.S. we're looking for more collaborators with software experience!)
- Please reference or write and reference an [issue](https://guides.github.com/features/issues/)
- Please have clear commit messages
- Please write detailed documentation and tests for every added piece of functionality
- Please try to not break existing functionality, or if you need to, please plan to justify this necessity and coordinate with the collaborators
- Please be patient and respectful with feedback
- Please use pre-commit hooks

14.4 Others

-Using pre-commit ` pip install pre-commit pre-commit install `

-Running Tests ``` -Locally python3 -m unittest discover

-Docker ./docker/bin/build_container.sh ./docker/bin/test.sh ```

CHAPTER

FIFTEEN

PUBLICATIONS

Papers by the Columbia research team can be found at Google Scholar.

Table 1: Publications

Title	Conference	Link	Citations	Year
FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance	NeurIPS 2021 Data-Centric AI Workshop	paper, code	2	2021
Explainable deep reinforcement learning for portfolio management: An empirical approach	ICAIF 2021: ACM International Conference on AI in Finance	paper, code	1	2021
FinRL-Podracer: High performance and scalable deep reinforcement learning for quantitative finance	ICAIF 2021: ACM International Conference on AI in Finance	paper, code	2	2021
FinRL: Deep reinforcement learning framework to automate trading in quantitative finance	ICAIF 2021: ACM International Conference on AI in Finance	paper, code	7	2021
FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance	NeurIPS 2020 Deep RL Workshop	paper, code	25	2020
Deep reinforcement learning for automated stock trading: An ensemble strategy	ICAIF 2020: ACM International Conference on AI in Finance	paper, code	44	2020
Multi-agent reinforcement learning for liquidation strategy analysis	ICML 2019 Workshop on AI in Finance: Applications and Infrastructure for Multi-Agent Learning	paper, code	19	2019
Practical deep reinforcement learning approach for stock trading	NeurIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services	paper, code	86	2018

CHAPTER

SIXTEEN

EXTERNAL SOURCES

The following contents are collected and referred by AI4Finance community during the development of FinRL and related projects. Some of them are educational and relatively easy while some others are professional and need advanced knowledge. We appreciate and respect the effort of all these contents' authors and developers.

16.1 Proof-of-concept

[1] FinRL: Deep Reinforcement Learning Framework to Automate Trading in Quantitative Finance Deep reinforcement learning framework to automate trading in quantitative finance, ACM International Conference on AI in Finance, ICAIF 2021.

[2] FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance A deep reinforcement learning library for automated stock trading in quantitative finance, Deep RL Workshop, NeurIPS 2020.

[3] Practical deep reinforcement learning approach for stock trading. NeurIPS Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy, 2018.

[4] Deep Reinforcement Learning for Trading. Zhang, Zihao, Stefan Zohren, and Stephen Roberts. The Journal of Financial Data Science 2, no. 2 (2020): 25-40.

[5] A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. Jiang, Zhengyao, Dixing Xu, and Jinjun Liang. arXiv preprint arXiv:1706.10059 (2017).

16.2 DRL Algorithms/Libraries

[1] Documentation of ElegentRL by AI4Finance Foundation.

[2] Spinning Up in Deep RL by OpenAI.

16.3 Theory

[1] Deep Reinforcement Learning: An Overview Li, Yuxi. arXiv preprint arXiv:1701.07274 (2017).

[2] Continuous-time mean–variance portfolio selection: A reinforcement learning framework. Mathematical Finance, 30(4), pp.1273-1308. Wang, H. and Zhou, X.Y., 2020.

[3] Mao Guan and Xiao-Yang Liu. Explainable deep reinforcement learning for portfolio man- agement: An empirical approach. ACM International Conference on AI in Finance, ICAIF 2021.

[4] ICAIF International Conference on AI in Finance.

16.4 Trading Strategies

[1] Deep reinforcement learning for automated stock trading: an ensemble strategy. ACM International Conference on AI in Finance, 2020.

[2] FinRL-Podracer: High performance and scalable deep reinforcement learning for quantitative finance. ACM International Conference on AI in Finance, ICAIF 2021.

[3] Multi-agent reinforcement learning for liquidation strategy analysis, paper and codes. Workshop on Applications and Infrastructure for Multi-Agent Learning, ICML 2019.

[4] Risk-Sensitive Reinforcement Learning: a Martingale Approach to Reward Uncertainty. International Conference on AI in Finance, ICAIF 2020.

[5] Cryptocurrency Trading Using Machine Learning. Journal of Risk and Financial Management, August 2020.

[6] Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation. Michaël Karpe, Jin Fang, Zhongyao Ma, Chen Wang. International Conference on AI in Finance (ICAIF'20), September 2020.

[7] Market Making via Reinforcement Learning. Thomas Spooner, John Fearnley, Rahul Savani, Andreas Koukorinis. AAMAS2018 Conference Proceedings

[8] Financial Trading as a Game: A Deep Reinforcement Learning Approach Huang, Chien Yi. arXiv preprint arXiv:1807.02787 (2018).

[9] Deep Hedging: Hedging Derivatives Under Generic Market Frictions Using Reinforcement Learning Buehler, Hans, Lukas Gonon, Josef Teichmann, Ben Wood, Baranidharan Mohan, and Jonathan Kochems. Swiss Finance Institute Research Paper 19-80 (2019).

16.5 Financial Big Data

[1] FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance. NeurIPS 2021 Data-Centric AI Workshop

16.6 Interpretation and Explainability

[1] Explainable Deep Reinforcement Learning for Portfolio Management: An Empirical Approach. Guan, M. and Liu, X.Y.. ACM International Conference on AI in Finance, 2021.

16.7 Tools or Softwares

[1] FinRL by AI4Finance Foundation.

[2] FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance, by AI4Finance Foundation.

[3] ElegantRL: a DRL library developed by AI4Finance Foundation.

[4] Stable-Baselines3: Reliable Reinforcement Learning Implementations.

16.8 Survey

[1] Recent Advances in Reinforcement Learning in Finance. Hambly, B., Xu, R. and Yang, H., 2021.

[2] Deep Reinforcement Learning for Trading—A Critical Survey. Adrian Millea, 2021.

[3] Modern Perspectives on Reinforcement Learning in Finance Kolm, Petter N. and Ritter, Gordon. The Journal of Machine Learning in Finance, Vol. 1, No. 1, 2020.

[4] Reinforcement Learning in Economics and Finance Charpentier, Arthur, Romuald Elie, and Carl Remlinger. Computational Economics (2021): 1-38.

[5] Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics Mosavi, Amirhosein, Yaser Faghan, Pedram Ghamisi, Puhong Duan, Sina Faizollahzadeh Ardabili, Ely Salwana, and Shahab S. Band. Mathematics 8, no. 10 (2020): 1640.

16.9 Education

[1] Coursera Overview of Advanced Methods of Reinforcement Learning in Finance. By Igor Halperin, at NYU.

[2] *Foundations of reinforcement learning with applications in finance* by Ashwin Rao, Tikhon Jelvis, Stanford University

CHAPTER

SEVENTEEN

FAQ

Version
0.3

Date
05-29-2022

Contributors
Roberto Fray da Silva, Xiao-Yang Liu, Ziyi Xia, Ming Zhu

This document contains the most frequently asked questions related to FinRL, which are based on questions posted on the slack channels and Github issues.

17.1 Outline

17.2 1-Inputs and datasets

- *Not yet. We're developing this functionality*
- *Not yet. We're developing this functionality*
- *Not yet. We're developing this functionality*
- *Not yet*
- *Yahoo Finance (through the yfinance library)*

- *Yahoo Finance (only up to last 7 days), through the yfinance library. It is the only option besides scraping (or paying for a service provider)*
- *No, as this is more of an execution strategy related to risk control. You can use it as part of your system, adding the risk control part as a separate component*
- *Yes, you can add it. Remember to check on the code that this additional feature is being fed to the model (state)*
- *No, you'll have to use a paid service or library/code to scrape news and obtain the sentiment from them (normally, using deep learning and NLP)*

17.3 2-Code and implementation

- *Yes, it does*
- *Yes, because the current parameters are defined for daily data. You'll have to tune the model for intraday trading*
- *Not many yet, but we're working on providing different reward functions and an easy way to set your own reward function*
- *Yes, but none is available at the moment. Sometimes in the literature you'll find this referred to as transfer learning*
- *Each model has its own hyperparameters, but the most important is the total_timesteps (think of it as epochs in a neural network: even if all the other hyperparameters are optimal, with few epochs the model will have a bad performance). The other important hyperparameters, in general, are: learning_rate, batch_size, ent_coef, buffer_size, policy, and reward scaling*
- *There are several, such as: Ray Tune and Optuna. You can start from our examples in the tutorials*
- *We suggest using ElegantRL or Stable Baselines 3. We tested the following models with success: A2C, A3C, DDPG, PPO, SAC, TD3, TRPO. You can also create your own algorithm, with an OpenAI Gym-style market environment*
-

Please update to latest version (https://github.com/AI4Finance-LLC/FinRL-Library), check if the hyperparameters used were not outside a normal range (ex: learning rate too high), and run the code again. If you still have problems, please check Section 2 (What to do when you experience problems)

- **raw-html**
  *What to do when you experience problems? *

1. Check if it is not already answered on this FAQ 2. Check if it is posted on the GitHub repo issues. If not, welcome to submit an issue on GitHub 3. Use the correct channel on the AI4Finance slack or Wechat group.*

- **raw-html**
  *Does anyone know if there is a trading environment for a single stock? There is one in the docs, but the collab link seems to be broken. *

  We did not update the single stock for long time. The performance for single stock is not very good, since the state space is too small so that the agent extract little information from the environment. Please use the multi stock environment, and after training only use the single stock to trade.

17.4 3-Model evaluation

-

  Not exactly. Depending on the period, the asset, the model chosen, and the hyperparameters used, BH may be very difficult to beat (it's almost never beaten on stocks/periods with low volatility and steady growth). Nevertheless, update the library and its dependencies (the github repo has the most recent version), and check the example notebook for the specific environment type (single, multi, portfolio optimization) to see if the code is running correctly

-

  We use the Pyfolio backtest library from Quantopian (https://github.com/quantopian/pyfolio), especially the simple tear sheet and its charts. In general, the most important metrics are: annual returns, cumulative returns, annual volatility, sharpe ratio, calmar ratio, stability, and max drawdown

-

  There are several metrics, but we recommend the following, as they are the most used in the market: annual returns, cumulative returns, annual volatility, sharpe ratio, calmar ratio, stability, and max drawdown

-

  We recommend using buy and hold (BH), as it is a strategy that can be followed on any market and tends to provide good results in the long run. You can also compare with other DRL models and trading strategies such as the minimum variance portfolio

17.5 4-Miscellaneous

-

*1. Read the documentation from the very beginning 2. Go through * `tutorials <https://github.com/AI4Finance-Foundation/FinRL/tree/master/tutorials> `_ *3. read our papers*

-

This is available on our Github repo https://github.com/AI4Finance-LLC/FinRL-Library

-

Participate on the slack channels, check the current issues and the roadmap, and help any way you can (sharing the library with others, testing the library of different markets/models/strategies, contributing with code development, etc)

-

Please read 1-Inputs and datasets

-

Please read 4-Miscellaneous

-

Please check our development roadmap at our Github repo: https://github.com/AI4Finance-LLC/FinRL-Library

-

FinRL aims for education and demonstration, while FinRL-Meta aims for building financial big data and a metaverse of data-driven financial RL.

17.6 5-Common issues/bugs

- **Package trading_calendars reports errors in Windows system:**
  Trading_calendars is not maintained now. It may report errors in Windows system (python>=3.7). These are two possible solutions: 1). Use python=3.6 environment. 2). Replace trading_calendars with exchange_caldenars.