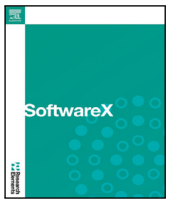
SoftwareX 30 (2025) 102106

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

gymfolio: A Reinforcement learning environment for Portfolio Optimization in Python

Francisco Espiga-Fernández *, Álvaro García-Sánchez, Joaquín Ordieres-Meré

Departamento de Ingeniería de Organización, Adm. de Empresas y Estadística, Escuela Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, José Gutiérrez Abascal, 2, 28006 Madrid, Spain

ARTICLE INFO

MSC:
68T05
91G10
91G80
62P05

Keywords:
Portfolio optimization
Reinforcement learning
Market simulation

ABSTRACT

This paper introduces `gymfolio`, a modular and flexible framework for portfolio optimization using reinforcement learning. `gymfolio` is built around the `PortfolioOptimizationEnv` class, enabling seamless integration of market observations, technical indicators, and dynamic rebalancing strategies. The implementation emphasizes adaptability, supporting extensions for custom investment goals and trading scenarios. `gymfolio` serves as a lightweight adaptable environment for advancing research in financial machine learning and promoting innovation in portfolio strategy development.

Code metadata

Current code version	0.1.2
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00036
Permanent link to Reproducible Capsule	For example: https://codeocean.com/capsule/9297207/tree/v1
Legal Code License	GPL-3
Code versioning system used	git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	python ≥ 3.10. Additional libraries can be found in Table 1
If available Link to developer documentation/manual	https://franespiga.github.io/gymfolio/
Support email for questions	francisco.espiga.fernandez@alumnos.upm.es

1. Motivation and significance

Portfolio optimization is fundamental in financial decision-making. Modern Portfolio Theory (MPT) [1] established the foundation for portfolio optimization by formalizing the relationship between risk and return. However, its reliance on static assumptions, such as normally distributed returns [2], limits its application in volatile market conditions. Reinforcement Learning (RL) surpasses these limitations by learning adaptive strategies that respond to evolving market dynamics [3–6].

`gymfolio` offers a modular RL framework tailored to portfolio optimization, allowing researchers to benchmark RL agents (e.g., Deep Q-Networks (DQN) [7], Proximal Policy Optimizaton (PPO) [8], Deep Deterministic Policy Gradient (DDPG) [9]) in diverse market scenarios.

As demonstrated in [10], `gymfolio` has been pivotal in comparing RL configurations with different agent architectures, environment signals, and historical depths. Notably, it enabled the identification of Convolutional Neural Networks (CNNs [11])-based models as highly effective for capturing dynamic market trends, achieving superior risk-adjusted returns.

Users can configure `gymfolio` with custom environments to align with specific investment goals and data inputs. Unlike alternatives like FinRL [12] and OR-Gym [13], `gymfolio` excels in managing multi-objective reward structures, incorporating advanced feature extractors, and mitigating trajectory correlation issues [14]. Innovations such as

* Corresponding author.
E-mail address: francisco.espiga.fernandez@alumnos.upm.es (Francisco Espiga-Fernández).

https://doi.org/10.1016/j.softx.2025.102106
Received 17 January 2025; Received in revised form 10 February 2025; Accepted 19 February 2025
Available online 6 March 2025

Table 1
gymfolio package dependencies and versions

Package	Version	Description
numpy	1.26.4	Fundamental package for array computing in Python
pandas	2.2.2	Powerful data structures for data analysis, time series, and statistics
gymnasium	0.29.1	Python library for developing and comparing RL algorithms.
stable-baselines3	2.3.2	Pytorch version of Stable Baselines, implementations of reinforcement learning algorithms.
torch	2.3.0	Tensors and Dynamic neural networks in Python with strong GPU acceleration
tqdm	4.66.4	Fast, Extensible Progress Meter
tables	3.9.2	Hierarchical datasets for Python

episodic instrument shifting and trajectory bootstrapping further enhance its robustness, making it particularly suited to non-stationary financial environments [10,15].

2. Software description

`gymfolio` serves as a dynamic framework tailored for RL-driven portfolio optimization. Leveraging the `Gymnasium` package, `gymfolio` ensures compatibility with modern RL libraries such as `Stable-Baselines3 (SB3)` [16], `PTAN`, and `RLLib` [17]. The lightweight architecture focuses on modularity and extensibility, relying on widely used libraries like `PyTorch` [18], `NumPy` [19], and `Pandas` [20], which simplifies adoption, compatibility and extension to other frameworks and libraries.

2.1. Software architecture

- `base.py`: Implements the foundational class `PortfolioOptimizationEnv`, establishing state transitions, action spaces, and a primary reward mechanism based on cumulative log returns between rebalancing periods.
- `common.py`: Provides essential utilities for preprocessing data and managing portfolio weights, ensuring data consistency and readiness for RL training workflows.
- `custom.py`: Introduces specialized classes like `SharpeEnv` that modifies the reward metric to the Sharpe ratio [21] for risk-reward maximization, `TrackingErrorEnv` for agents trying to mimic a reference index using a subset of investment instruments, and `CompositeRewardEnv` for multi-objective optimization using a pseudo-IRL approach. These environments inherit their base methods from `PortfolioOptimizationEnv`.
- `metrics.py`: Offers a comprehensive suite of financial metrics, including Sharpe [21], Calmar [22], and Sortino [23] ratios, for evaluating portfolio performance (see Fig. 1).

2.2. Software functionalities

`gymfolio` provides a comprehensive set of functionalities to enable effective portfolio optimization using RL. Its design emphasizes flexibility, usability, and extensibility to accommodate a broad spectrum of research and practical applications.

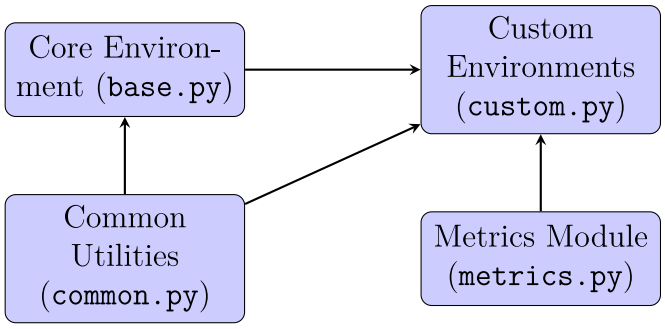


Fig. 1. gymfolio modules.

2.2.1. Customization

`gymfolio` allows users to customize key parameters to align the environment with specific investment strategies and market conditions. For instance:

- `rebalance_every`: Controls the frequency of portfolio rebalancing actions, enabling experimentation with short-term and long-term strategies.
- `observation_frame_lookback`: Specifies the historical depth of market signals considered by the agent.
- Support for `discrete` and `continuous` action spaces: Allows users to select between granular -diversified portfolio with fractional weight changes- or high-level -full portfolio allocation to a single instrument- portfolio adjustments.
- Transaction costs and slippage models: Ensures realistic simulations by incorporating real-world constraints such as market impact and execution costs.

These features enable researchers to experiment with various configurations, tailoring `gymfolio` to meet a wide range of objectives and constraints.

2.2.2. Observations

The observation space in `gymfolio` is highly flexible, supporting three distinct modes tailored for different machine learning models:

- `Vector mode`: Collapses the observation space into a single-dimensional array, compatible with multi-layer perceptrons and simple decision trees.
- `Tile mode`: Retains a two-dimensional structure (lookback x indicators), ideal for sequential models like recurrent neural networks (RNNs [24]).
- `Tensor mode`: Constructs a 3D tensor (channels x height x width), incorporating per-instrument and global indicators. This mode is optimized for CNNs [11] and supports sophisticated feature extraction.

This versatility enables `gymfolio` to seamlessly integrate with various RL algorithms and adapt to the unique requirements of various research scenarios, as shown in Fig. 2.

2.2.3. Rewards

`gymfolio` includes several pre-defined reward functions and supports the creation of custom rewards, fostering experimentation with multi-objective optimization. Key reward mechanisms include

- `SharpeEnv`: Computes rewards based on the Sharpe ratio, balancing risk and return.
- `TrackingErrorEnv`: Focuses on minimizing the deviation from a reference index.

- `CompositeRewardEnv`: Implements a pseudo-inverse reinforcement learning (IRL) approach, enabling optimization across multiple objectives such as risk-reward metrics or other user defined constraints.

The modular reward structure encourages users to develop custom metrics tailored to their specific research needs, enhancing `gymfolio`'s versatility.

3. Illustrative examples

To demonstrate the capabilities of `gymfolio`, we present a reproducible example using the RL environment provided in `base.py`, combined with evaluation metrics of `metrics.py` and utility functions of `common.py`. The example employs a PPO agent from SB3 to optimize a portfolio in a simulated financial market.

3.1. Experimental setup

This example uses two primary datasets:

- **Market Data (`df_ohlc`)**: Open, High, Low, Close (OHLC) historical prices at daily granularity for the S&P500 index. Table 2 provides a sample of the OHLC dataset.
- **Technical Indicators (`df_observations`)**: Timestamped data including moving averages, prices, and technical indicators such as Relative Strength Index (RSI) [25], Average True Range (ATR) [26].

3.2. Environment configuration

We configure the `gymfolio` environment using the following settings:

- `df_ohlc`: Price data for each investment vehicle.
- `df_observations`: prices, moving averages and technical indicators (RSI, ATR, etc.).
- `rebalance_every=5`: Weekly portfolio adjustments.
- `max_trajectory_len=252`: Simulates approximately 252 trading days (1 year).
- `observation_frame_lookback=3`: Rolling window of three historical observations.
- `continuous_weights=True`: Continuous allocation across assets.

The environment settings provide flexibility to model different trading strategies, enabling experimentation with rebalancing frequencies, observation windows, and action spaces.

```
from gymfolio.envs.base import PortfolioOptimizationEnv
from stable_baselines3 import PPO

env = PortfolioOptimizationEnv(
  df_ohlc=data.dropna(),
  df_observations=indicators.dropna(),
  rebalance_every=5,
  max_trajectory_len=252,
  observation_frame_lookback=3,
  continuous_weights=True,
  verbose=2
)
model = PPO('MlpPolicy', env, batch_size=256)
```

Table 2
Sample OHLC data from the S&P500 index.

OHLC Data (^GSPC - S&P500)				
Timestamp	Open	High	Low	Close
2023-01-04	3840.36	3873.15	3815.77	3852.97
2023-01-05	3839.73	3839.73	3802.41	3808.10
2023-01-06	3823.37	3906.18	3809.56	3895.08

3.3. Agent training

The agent interacts with the environment over a sequence of time steps. At each step, it receives observations, predicts actions (portfolio weights), and evaluates rewards. In each rebalancing step, the agent observes the `df_observations` signals, computes actions, and receives rewards based on portfolio performance.

```
model.learn(total_timesteps=10_000)
model.save('/.final_model')
```

Training stops after 10,000 steps (equivalent to 40 simulated years), and the model is saved for future use.

3.4. Agent interaction

The `env.reset()` method initializes the environment at a random trading day.

1. Receives an observation from the environment. Prior to transforming the observation in vector, tile or tensor, it is a `pd.DataFrame` of shape `(observation_frame_lookback, df_observations.shape[1])`
2. Processes the observation and outputs the action, a vector of shape $[1, n\_instruments]$ with the portfolio weights $\vec{w}$.
3. The action is passed to the environment, returning the next observation, computed reward, and `done` flag.

```
obs = env.reset()
action, _ = model.predict(obs)
obs, reward, done, info = env.step(action)
```

3.5. Evaluation metrics

`gymfolio` computes rewards using metrics like the Sharpe ratio, tracking error, and cumulative returns. For this example:

$$R_t = \log(1 + \vec{w} \cdot \vec{r}), \tag{1}$$

where $\vec{w}$ represents portfolio weights and $\vec{r}$ represents returns for the rebalancing period.

3.5.1. Benchmarks

The evaluation loop compares the agent portfolio against benchmarks like the market and equal-weight portfolios and individual buy-and-hold strategies. Fig. 3 shows cumulative returns over the testing period.

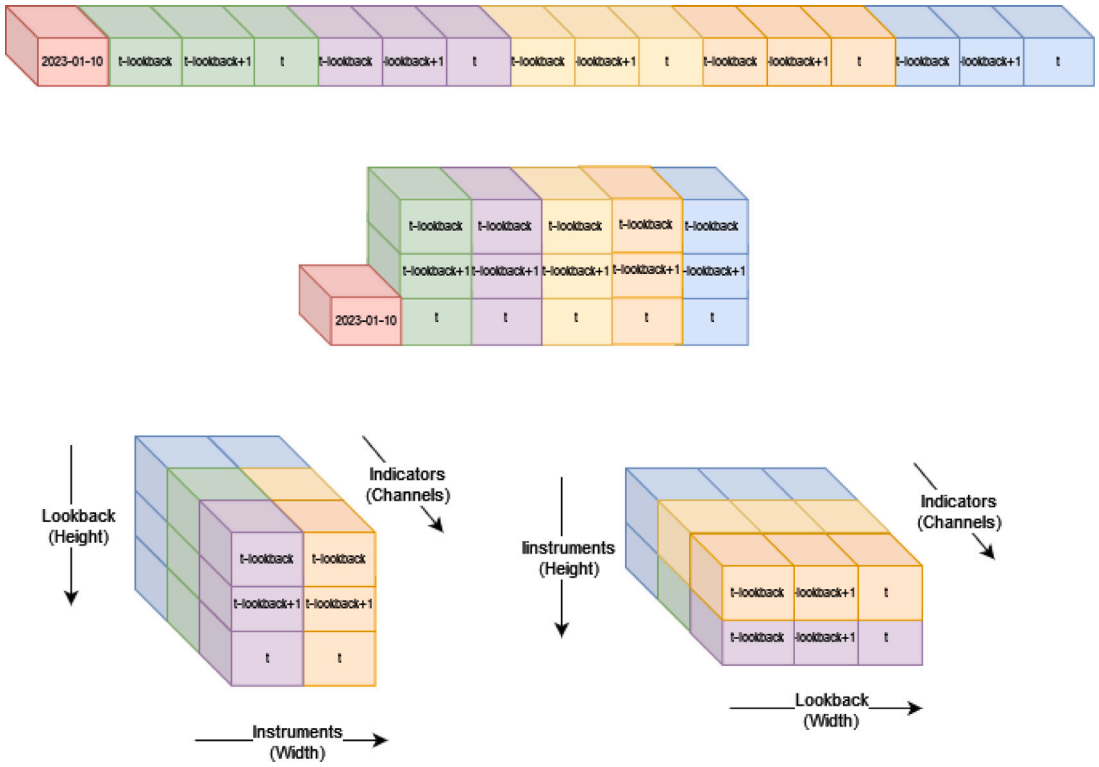


Fig. 2. Example configurations of vector, tile and tensor observation modes in gymfolio.

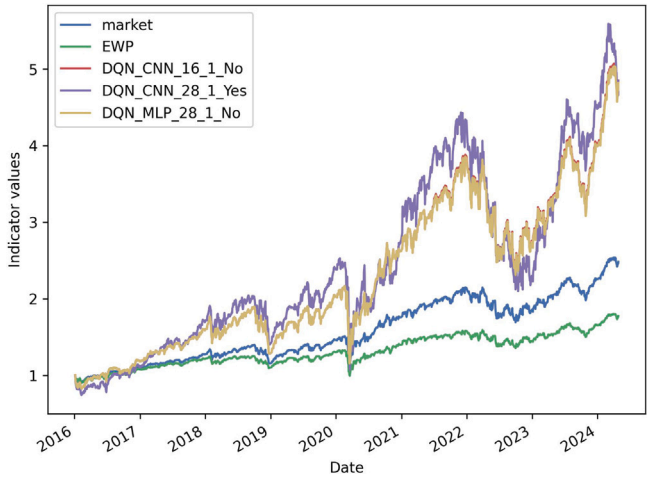


Fig. 3. Cumulative returns of the trained agents in [10], the market and the equal weights portfolio.

3.6. Reproducibility

This example is fully reproducible using publicly available data [27] and code in the `gymfolio` repository [28]. Researchers can adapt the provided script to test other RL agents or customize environment parameters. Table A.3 provides a detailed description of the parameters of the base environment, and Table A.4 of the parameters of the custom environments. For further customization, consult the official `gymfolio` documentation [29].

4. Impact

Portfolio optimization remains a cornerstone in financial management, and advancements in RL methodologies promise significant contributions to this field. `gymfolio` introduces novel features, including trajectory bootstrapping, episodic instrument shifting, and a pseudo-IRL mechanism, addressing key challenges in non-stationary markets and correlated trajectories. In the following, we discuss its specific impacts.

4.1. New research questions enabled

The design of `gymfolio` allows researchers to systematically explore new configurations of RL algorithms, feature extractors, and reward structures in portfolio optimization. This includes investigations into optimal rebalancing strategies, the role of multi-objective reward functions, and the use of advanced feature extraction techniques, such as CNNs and LSTMs. For example, trajectory bootstrapping provides an innovative way to analyze the effects of market non-stationarity on learning stability.

4.2. Enhancements to existing research questions

The framework has already been utilized to benchmark RL agent configurations in terms of history depth, rebalancing frequency, and feature extractor efficacy [10]. This systematic comparison highlights `gymfolio`'s ability to advance research into RL applications in financial markets by enabling fair and reproducible evaluations. The use of episodic instrument shifting and composite reward mechanisms further refines the ability of RL agents to adapt dynamically to market volatility.

4.3. Impact on daily practices

Practitioners in asset management can leverage `gymfolio`'s customizable environment to develop and test bespoke investment strategies. By supporting a wide range of configurations, from discrete to continuous action spaces and various observation formats, `gymfolio` facilitates rapid prototyping and validation of trading strategies under realistic market conditions. This has the potential to streamline the strategy design process in quantitative finance.

4.4. Adoption and usage

`gymfolio` is currently in its early stages of adoption in both academic and industry settings. [10] demonstrates its application in systematic RL benchmarking, underscoring its role as a foundational tool in financial machine learning. Continued dissemination through open source contributions and academic publications is expected to expand its user base.

5. Conclusions

`gymfolio` provides a robust and flexible framework for the optimization of portfolios based on reinforcement learning. By incorporating innovative features such as trajectory bootstrapping, episodic instrument shifting, and composite reward mechanisms, it addresses critical challenges like market non-stationarity and correlated observations, enabling the development of more robust RL agents.

The modular architecture of the framework, the compatibility with leading RL libraries, and the support for various reward structures make it a valuable tool for both academic research and practical applications. `gymfolio` has already demonstrated its impact through systematic benchmarking studies [10] and is well positioned to drive future advances in automated financial decision-making.

By lowering the barriers to entry for RL experimentation in finance, `gymfolio` promotes reproducibility, innovation, and collaboration between disciplines. Future developments will focus on expanding its functionality to support additional RL models such as Decision Transformers and alternative reward objectives. Furthermore, `gymfolio` currently lacks direct support for ingestion of real-time market data. Incorporating mechanisms for seamless integration with live financial data sources would enhance its applicability to real-world trading scenarios, enabling more adaptive and responsive portfolio optimization strategies.

CRediT authorship contribution statement

Francisco Espiga-Fernández: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Álvaro García-Sánchez:** Writing – review & editing, Validation, Supervision, Methodology. **Joaquín Ordieres-Meré:** Writing – review & editing, Validation, Supervision, Funding acquisition.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used Writefull and ChatGPT to improve the readability and language of this manuscript. After using this tool and service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Joaquin Ordieres Mere reports article publishing charges was provided by Ministerio de Ciencia, Innovación y Universidades of Spain. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors want to thank the Spanish Agencia Estatal de Investigación because this research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (Grant Ref. PID2022-137748OB-C31 funded by MCIN/AEI/10.13039/501100011033) and ''ERDF A way of making Europe''. The authors want to thank Diego Serrano Venturini for his suggestions regarding modularity and testing throughout the design and implementation of `gymfolio`.

Table A.3
Overview of the parameters of the PortfolioOptimizationEnv base environment.

Name	Type	Default	Description
df_ohlc	pd.DataFrame	–	OHLC of the portfolio instruments
df_observations	pd.DataFrame	–	environment features
rebalance_every	int	1	periods between consecutive rebalancing actions
slippage	float	0.0005	%loss due to gap between decision price for the agent and the execution price
transaction_costs	float	0.0002	%loss due to execution of the trade
continuous_weights	bool	False	'False' to split the weights in (h)eld, (b)ought and (s)old positions
allow_short_positions	bool	False	'True' to enable short positions
max_trajectory_len	int	252	max total number of periods for the trajectories E.g. 252 for a trading year
observation_frame_lookback	int	16	return the previous N observations from the environment to take the next action
render_mode	str	'tile'	Either 'tile' (2D), 'tensor'(3D) or 'vector'(1D) to return the environment state
agent_type	str	'discrete'	'discrete' or 'continuous' action space
convert_to_terminated_truncated	bool	'True'	use 'done' (old `Gym` version) or 'truncated' and 'terminated' (new `Gymnasium` version)

Table A.4
Additional parameters of the SharpeEnv and TrackingErrorEnv.

Name	Type	Default	Description
df_reference	pd.DataFrame	–	DataFrame with the reference returns of the tracked index or instrument for the `TrackingErrorEnv`
riskfree_rate	float	0.0	Risk-free rate to compute *Sharpe* ratio
periods_per_year	int	252	Number of periods per year to annualize the returns
compute_cumulative	bool	False	'False' to consider all the episode trajectory to compute the ratios and 'True' to consider between rebalancing periods on the `SharpeEnv`

Appendix A. Environment parameters

See Tables A.3 and A.4.

Appendix B. Overview of the datasets used in Section 3

This appendix describes the datasets contained in the HDF5 file `example.h5`, which are utilized in the illustrative example. The file includes the following datasets:

- **instruments**: Contains market data such as open, high, low, and close prices for various financial instruments.
  - Columns: (35) OHLC and volume columns for instruments ^RUT, ^IXIC, ^GDAXI, ^FTSE, ^N225, ^GSPC and CASH.
  - Number of observations: 24695
  - Time range: 1927-12-30 to 2024-05-02.
- **technical_indicators**: Contains calculated technical indicators relevant for state representation.
  - Columns: (6) Normalized Average True Range (NATR) [26] for instruments ^RUT, ^IXIC, ^GDAXI, ^FTSE, ^N225, ^GSPC.
  - Number of observations: 8396
  - Time range: 1988-01-05 to 2024-04-30.

The data sets can be accessed in Python using the following code:

```python
import pandas as pd

# Path to the HDF5 file
file_path = "example.h5"

# Reading the 'instruments' dataset
instruments = pd.read_hdf(file_path, 'instruments')

# Reading the 'technical_indicators' dataset
indicators = pd.read_hdf(file_path, 'technical_indicators')
```

Appendix C. Code deep-dive

This appendix contains the most relevant code snippets of the sequence of steps followed in the example of Section 3.

```
for idx, row in eval_data.dropna().tail(
 eval_data.dropna().shape[0] - lookback).iterrows():
  if action is not None:
    if agent_types[agent_algorithm]=='discrete':
      actions = np.zeros(len(instruments))
      actions[int(action[0])] = 1.0
      different_actions.append(int(action[0]))
      decision_series[idx] = actions
    else:
      decision_series[idx] = action[0]
  observation_frame = eval_data.loc[:idx, :].tail(lookback)

  if observation_frame.shape[0] < lookback:
    break
  action = model.predict(
   torch.Tensor(observation_frame.values),
   deterministic=True)
```

Listing 1: Agent evaluation using a `pd.DataFrame`

```
eval_data = pd.concat([train_indicators, test_indicators], axis = 0)
eval_env = PortfolioOptimizationEnv(
  df_ohlc=data.dropna(),
  df_observations=eval_data.dropna(),
  rebalance_every=5,
  max_trajectory_len=eval_data.shape[0],
  observation_frame_lookback=3,
  continuous_weights=True,
  verbose=2
)

done = False
action = None
observation_frame = pd.DataFrame(eval_env.reset())
decision_series = {}

while not done:
  idx = eval_env.current_rebalancing_date
  if action is not None:
    # Take the best action (investment instrument)
    # in discrete agents
    if agent_types[agent_algorithm]=='discrete':
      actions = np.zeros(len(instruments))
      actions[int(action[0])] = 1.0
      different_actions.append(int(action[0]))
      decision_series[idx] = actions
    # In continuous agents, diversify the portfolio
    else:
      decision_series[idx] = action[0]

  if observation_frame.shape[0] < lookback:
    break
  # deterministic = True to ensure no exploration
  # in the agent is taken.
  action, _ = model.predict(
   torch.Tensor(observation_frame.values),
   deterministic=True)
  observation_frame, reward, done, info = eval_env.step(action)
```

Listing 2: Agent evaluation using a `gymfolio.env`

```
R_h, R_b, R_s = create_return_matrices(data)
R_h.index = pd.to_datetime(R_h.index)
R_b.index = pd.to_datetime(R_b.index)
R_s.index = pd.to_datetime(R_s.index)

missing_indices = set(df_weights.index)-set(data.index)
logger.warning(f"Missing indices: {list(missing_indices)}")
missing_indices = set(data.index)-set(df_weights.index)
logger.warning(f"Missing indices: {list(missing_indices)}")

current_weights = np.zeros(len(instruments))
returns = {}
for idx, row in tqdm(df_weights.iterrows()):
  if idx in R_h.index:
    new_weights = row.values
    w_h, w_b, w_s = decompose_weights_tensor(
      torch.Tensor(new_weights), torch.Tensor(current_weights))
    r_h = torch.Tensor(R_h.loc[idx])
    r_b = torch.Tensor(R_b.loc[idx])
    r_s = torch.Tensor(R_s.loc[idx])

    pf_h = torch.dot(w_h, r_h)
    pf_b = torch.dot(w_b, r_b)
    pf_s = torch.dot(w_s, r_s)

    pf_rets = pf_h + pf_b + pf_s

    returns[idx] = pf_rets.detach().numpy().tolist()

    current_weights = new_weights

df_strategy = pd.DataFrame.from_dict(returns,
                    orient = 'index',
                    columns = ['strategy_returns'])
df_strategy.index = pd.to_datetime(df_strategy.index)
```

Listing 3: Weights and returns decomposition

References

[1] Markowitz H. Portfolio selection. J Financ 1952;7(1):77–91. http://dx.doi.org/10.1111/j.1540-6261.1952.tb01525.x.

[2] Merton RC, Samuelson PA. Fallacy of the log-normal approximation to optimal portfolio decision-making over many periods. J Financ Econ 1974;1(1):67–94. http://dx.doi.org/10.1016/0304-405X(74)90009-9.

[3] Cong LW, Tang K, Wang J, Zhang Y. AlphaPortfolio for investment and economically interpretable AI. In: Proceedings of the AAAI conference on artificial intelligence. 2020, p. 1–76, URL https://api.semanticscholar.org/CorpusID:219127209.

[4] Benhamou E, Saltiel D, Ohana J, Atif J, Laraki R. Deep reinforcement learning (DRL) for portfolio allocation. In: Machine learning and knowledge discovery in databases. Applied data science and demo track. Springer Cham; 2021, p. 527–31. http://dx.doi.org/10.1007/978-3-030-67670-4_32.

[5] Odermatt L, Beqiraj J, Osterrieder J. Deep reinforcement learning for finance and the efficient market hypothesis. SSRN Electron J 2021. http://dx.doi.org/10.2139/ssrn.3865019.

[6] Benhamou E, Saltiel D, Ungari S, Mukhopadhyay A. Bridging the gap between markowitz planning and deep reinforcement learning. 2020, arXiv:2010.09108.

[7] Mnih, et al. Human-level control through deep reinforcement learning. Nature 2015;518:529–33. http://dx.doi.org/10.1038/nature14236.

[8] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv:1707.06347.

[9] Lillicrap TP, et al. Continuous control with deep reinforcement learning. 2019, arXiv:1509.02971.

[10] Espiga-Fernández F, García-Sánchez Á, Ordieres-Meré' J. A systematic approach to portfolio optimization: A comparative study of reinforcement learning agents, market signals, and investment horizons. Algorithms 2024;17(12). http://dx.doi.org/10.3390/a17120570, URL https://www.mdpi.com/1999-4893/17/12/570.

[11] LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series. In: The handbook of brain theory and neural networks. Cambridge, MA, USA: MIT Press; 1998, p. 255–8.

[12] Liu X-Y, et al. FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. 2022, arXiv:2011.09607.

[13] Hubbs CD, Perez HD, Sarwar O, Sahinidis NV, Grossmann IE, Wassick JM. OR-gym: A reinforcement learning library for operations research problems. 2020, arXiv:2008.06319.

[14] Hessel M, et al. Rainbow: Combining improvements in deep reinforcement learning. 2017, arXiv:1710.02298.

[15] Velay M, Doan B-L, Rimmel A, Popineau F, Daniel F. Benchmarking robustness of deep reinforcement learning approaches to online portfolio management. In: 2023 international conference on innovations in intelligent systems and applications. IEEE; 2023, p. 1–6. http://dx.doi.org/10.1109/inista59065.2023.10310402.

[16] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable reinforcement learning implementations. J Mach Learn Res 2021;22(268):1–8, URL http://jmlr.org/papers/v22/20-1364.html.

[17] Liang E, et al. RLlib: Abstractions for distributed reinforcement learning. 2018, arXiv:1712.09381.

[18] Paszke A, et al. PyTorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems 32. Curran Associates, Inc.; 2019, p. 8024–35, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[19] Harris CR, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[20] pandas development team T. pandas-dev/pandas: Pandas. Zenodo; 2020, http://dx.doi.org/10.5281/zenodo.3509134.

[21] Sharpe W. Mutual fund performance. J Bus 1965;39. URL https://EconPapers.repec.org/RePEc:ucp:jnlbus:v:39:y:1965:p:119.

[22] Young T. Calmar ratio: A smoother tool. Futures 1991;20:40–1. http://dx.doi.org/10.3905/jpm.1991.4093439.

[23] Sortino FA, van der Meer R. Downside risk. J Portf Manag 1991;17:27–31. http://dx.doi.org/10.3905/jpm.1991.4093439.

[24] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9:1735–80. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[25] Wilder J. New concepts in technical trading systems. Trend Research; 1978, p. 63–70.

[26] Wilder J. New concepts in technical trading systems. Trend Research; 1978, p. 21–3.

[27] Espiga-Fernandez F. RL Package dataset for example ex_SB3.py. 2024, http://dx.doi.org/10.6084/m9.figshare.27325329.v1, https://figshare.com/articles/dataset/RL_Package_dataset_for_example_ex_SB3_py/27325329. [Accessed: 2024-12-20].

[28] Espiga-Fernandez F. Gymfolio. Zenodo; 2024, http://dx.doi.org/10.5281/zenodo.14551666.

[29] Gymfolio documentation. 2024, URL https://franespiga.github.io/gymfolio/.