КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформатика»
спеціальності 122 «Комп'ютерні науки» на тему:

ІНТЕЛЕКТУАЛЬНА СТРАТЕГІЯ ТОРГІВЛІ ІЗ ВИКОРИСТАННЯМ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Виконав студент 4-го курсу
Винник Дмитро Олександрович _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Панченко Тарас Володимирович _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«_____»_____ 2021 р.,
протокол №_____
Завідувач кафедри
Нікітченко М.С. _____
(підпис)

Київ – 2021

ЗМІСТ

РЕФЕРАТ

Бакалаврська робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел (28 найменувань) та 1 додатку. Робота містить 15 рисунків та 5 таблиць. Загальний обсяг роботи становить 32 сторінки, основний текст роботи викладено на 20 сторінках.

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, СЕРЕДОВИЩЕ, АГЕНТ, Ф'ЮЧЕРСИ, ПОКАЗНИКИ БІРЖ, ДРОБОВЕ ДИФЕРЕНЦІЮВАННЯ, Q-НАВЧАННЯ, DQN.

Об'єктом дослідження є ринок цінних паперів, зокрема криптовалютні біржі. Предметом дослідження є алгоритмічна торгівля, що базується на внутрішніх та зовнішніх показниках.

Метою роботи є перевірка гіпотези про прибутковість RL-стратегії за наявності широкого спектра даних, на великому часовому інтервалі, враховуючи комісію, заокруглення та зсуви ціни.

Методи розроблення: штучні нейронні мережі, навчання з підкріпленням, дробове диференціювання, тести на стаціонарність та одиничний корінь. Інструменти розроблення: середовище розробки PyCharm, мова програмування Python 3.

Результати роботи: проаналізовані доступні в аналітичних сервісах дані, виділені корисні показники, вивантажено набір даних, реалізовано та досліджено швидке дробове диференціювання, стандартизовано дані, визначено архітектуру нейронної мережі та гіперпараметри, розроблено середовище для навчання агентів, протестовано збіжність і дохідність агенту до та після введення обмежень.

СКОРОЧЕННЯ

RL – Reinforcement Learning
Навчання з підкріпленням

ADF-GLS – Augmented Dickey-Fuller test with Generalized Least Squares
Розширений тест Дікі-Фуллера
з узагальненим методом найменших квадратів

KPSS – Kwiatkowski–Phillips–Schmidt–Shin
Тест Квятковського, Філліпса, Шмідта, Шина

ReLU – Rectified linear unit
Випрямлений лінійний вузол

AR(F)IMA – Autoregressive (fractionally) integrated moving average
Авторегресивне (дробове) інтегроване рухоме середнє

VE – Value error
Відхилення цінності

(I)FFT – (Inverse) fast Fourier transform
(Обернене) швидке перетворення Фур'є

API – Application programming interface
Прикладний програмний інтерфейс

ВСТУП

Оцінка сучасного стану об’єкта розробки

RL існує ще з кінця минулого століття, тому робіт пов’язаних з торгівлею акціями досить багато, натомість торгівля криптовалютами набула популярності лише в середині 2010-тих років, тому робіт мало і роботи сильно відрізняються підходами, даними, використаними обмеженнями чи навіть цілями: метою одної роботи може бути торгівля одним активом, метою іншої – формування портфеля активів.

Актуальність роботи та підстави для її виконання

Криптовалютні біржі відкрили можливість для спекуляцій широкому загалу. Для роботи з ними не потрібні посередники (брокери), бо на них можна торгувати мільйонними частинами активу, а позикові гроші для коротких позицій біржа надає автоматично. Значна частина активів на біржах є високоліквідною, що дозволяє часто виставляти заявки, які з високою імовірністю будуть виконані, а отже можливою є як торгівля в довгостроковій перспективі, так і високочастотна торгівля.

Для торгівлі треба приймати рішення. Це можна робити випадковим чином, чи за “фігурами” на графіку, чи за положенням зірок на небі. Головним є те, чи приносить це прибуток. Практика показує, що для прийняття рішень можна ефективно використовувати різні внутрішні й зовнішні показники. Наприклад до внутрішніх можна віднести об’єм торгів, а до зовнішніх – кількість видобутих майнерами монет.

Фактично дані є багатовимірним часовим рядом. Це може схиляти до використання методів прогнозування, але ця проміжна ланка між даними та рішенням погіршує точність моделі. У цій ситуації доцільніше визначати виконувану дію безпосередньо з даних.

Основною перешкодою для використання навчання з учителем, тобто класифікації та регресії, є маркування даних, бо без нього неможливо визначити цільову функцію: ми могли б побудувати дерево прийняття рішень – класифікатор, що ставить у відповідність станам оптимальні дії, але для цього

потрібно знати, чи мати змогу визначити, які дії є оптимальними. Навчання без учителя, яке так чи інакше виділяє з даних патерни (кластери, компоненти, карти), теж не є застосовним, бо не зрозуміло як перетворити інформацію про патерни з невідомими властивостями в оптимальну дію. Тут в пригоді стає навчання з підкріпленням (RL).

RL враховує в оцінці стану або пари (стан, дія) майбутні винагороди. Також, завдяки Навчанню тимчасовим різницям (TD) можливим є навчання після кожного рішення. Тому нагальною є перевірка саме здібностей RL.

Мета та завдання роботи

Метою роботи є перевірка гіпотези про прибутковість RL-стратегії за наявності широкого спектра даних, на великому часовому інтервалі, враховуючи комісію, заокруглення та зсуви ціни. Для досягнення цієї мети поставлено такі завдання:

- проаналізувати доступні в аналітичних сервісах дані
- виділити корисні показники
- завантажити дані
- стандартизувати дані
- визначити архітектуру нейронної мережі та гіперпараметри
- розробити середовище для навчання агентів
- протестувати збіжність і дохідність агенту до та після введення обмежень

Об’єкт, методи й засоби дослідження та розроблення

Біржу, як об’єкт, мімікрує розроблене OpenAI gym [1] середовище для навчання агентів. У якості агента виступає DQN [2] [3]. Програмну реалізацію вивантаження та обробки даних, агента та середовища виконано мовою Python 3 із використанням таких бібліотек як: TensorFlow [4], Keras [5], keras-rl [6], gym [1], NumPy [7], SciPy [8], arch [9], scikit-learn [10], pandas [11], matplotlib [12].

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ НАВЧАННЯ З ПІДКРІПЛЕННЯМ [13]

1.1. Задача *k*-рукого «Бандита»

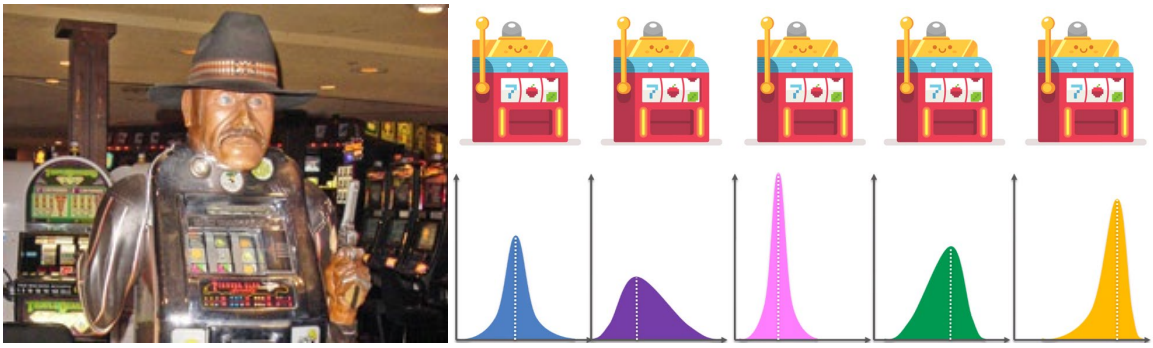

Рисунок 1 – Ілюстрація до задачі

«Бандит» – це слотовий автомат у вигляді бандита. У слот автомата кладуть монету та тягнуть за руку, в результаті отримують виграш.

Задача *k*-рукого «Бандита» це математичне уособлення реальної задачі максимізації винагород, де ми маємо k таких автоматів, кожен з яких має свій сталий розподіл винагород. На кожному кроці ми маємо вибирати автомат, не знаючи його розподіл.

Дохідність дії оцінюють наступним табличним методом:

A_t — (action) дія в момент t, скаляр

R_t — (reward) винагорода в момент t, скаляр

$Q_t(a)$ — цінність

$q_*(a) = \mathbb{E}[R_t|A_t = a]$ — істинна цінність

$$Q_t(a) \doteq \frac{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a} \cdot R_i}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} = \frac{\text{сума винагород коли дія } a \text{ вибрана до моменту } t}{\text{кількість разів коли дія } a \text{ вибрана до моменту } t}$$

Таблиця 1 – Приклад обчислення $Q_t(x)$, коли $A = \{x, y\}$

$A_1 = x,\ R_1 = 2$	$A_2 = y,\ R_2 = 0$	$A_3 = x,\ R_3 = 4$	$A_4 = y,\ R_4 = 1$	…
$Q_1(x) = 0$	$Q_2(x) = \frac{2}{1} = 2$	$Q_3(x) = Q_2(x) = 2$	$Q_4(x) = \frac{2+4}{2} = 3$	…

Тоді, знаючи цінність різних дій, залишилось вибрати дію з найбільшою цінністю. Цей підхід називають жадібним:

$$A_t = \underset{a}{\operatorname{argmax}}\, Q_t(a)$$

Але у ньому відсутнє розвідування, тому використовують ε-жадібне вибирання:

$$A_t \overset{\text{def}}{=} \begin{cases} \underset{a}{\operatorname{argmax}}\, Q_t(a), & \text{з імовірністю } 1 - \varepsilon \\ \text{випадково вибрана дія}, & \text{з імовірністю } \varepsilon \end{cases}$$

По суті ми визначаємо середню винагороду для кожної дії. Обраховувати середнє можна ітеративно:

$$Q_{n+1} = \bar{R}_n = \frac{R_1 + R_2 + \cdots + R_n}{n} = Q_n + \frac{1}{n}(R_n - Q_n)$$

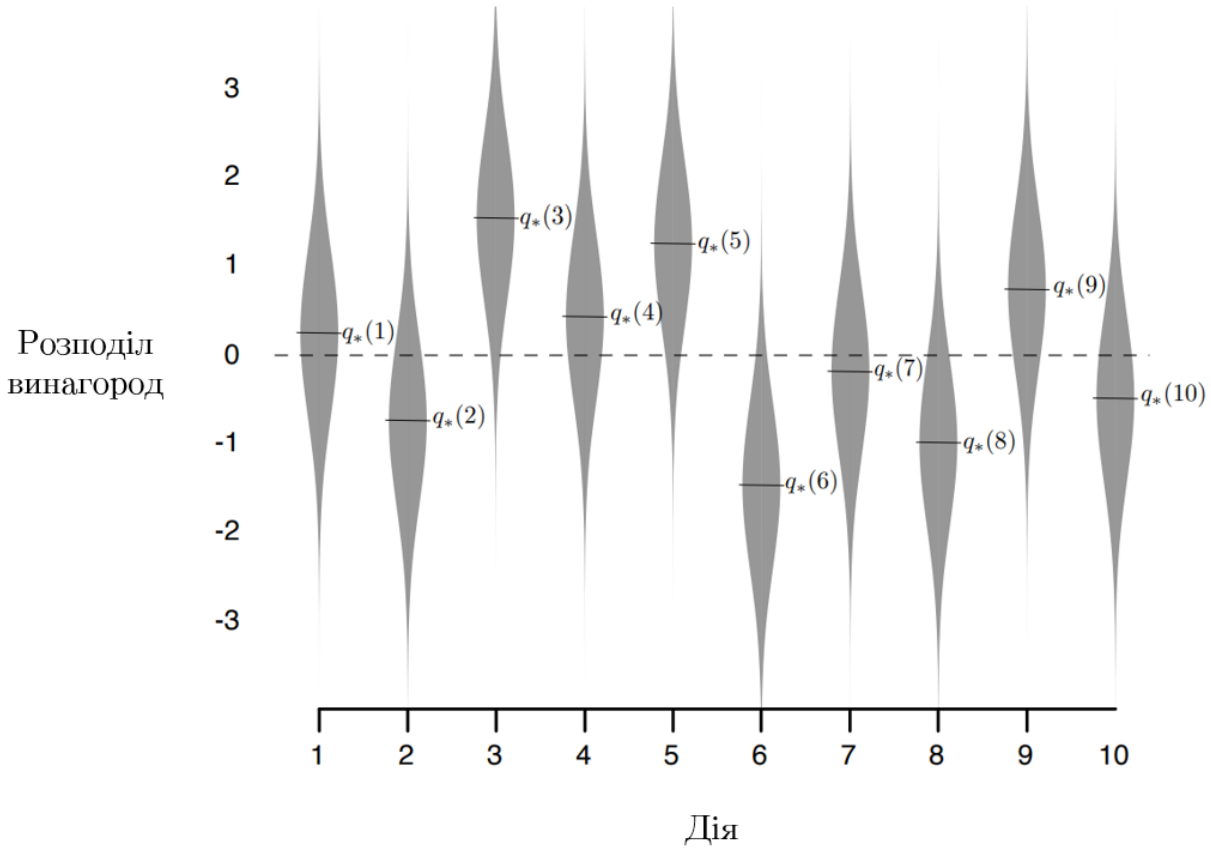

Рисунок 2 – Приклад «10-рукого Бандита»[1]. Оптимальною є третя дія

[1] Середнє збігається із піковим значенням щільності, що у загальному випадку є вірним тільки для унімодальних симетричних розподілів.

1.2. Нестаціонарні задачі

Задачі в яких цінність дій змінюється із плином часу називають нестаціонарними. Для таких задач функція цінності має поступово зменшувати вплив старих винагород. Для цього множник перед дужками замінюють на константне значення $\alpha \in (0,1)$:

$$Q_{n+1} := Q_n + \frac{1}{n}(R_n - Q_n) \;\rightarrow\; Q_{n+1} := Q_n + \alpha(R_n - Q_n)$$

Фактично, це експоненційне рухоме середнє.

1.3. Задачі зі станами

$S - (state)$ стан, арифметичний вектор

$\pi_t(a) = \pi(a|S_t) -$ стратегія, імовірність вибрати дію a у стані s

$Q_t = Q(S_t, A_t) -$ цінність дії

$V_t = V(S_t) = \pi_t Q_t -$ цінність стану

$S_0,\ A_0,\ R_1,\ \ldots,\ S_{T-1}, A_{T-1}, R_T -$ траєкторія

У задачах зі станами кожна дія супроводжується винагородою і переходом у новий стан, тому наші уявлення про очікуваний дохід змінюються, бо очікуваний дохід має включати не тільки миттєві, а і майбутні винагороди. Через це винагороду R_t замінюють на сукупну винагороду $G_t = R_{t+1} + R_{t+2} + \cdots$, що рекурсивно виглядає як $G_t = R_{t+1} + G_{t+1}$. Також додають знецінювальний коефіцієнт $\gamma \in (0,1)$ для зменшення впливу майбутніх винагород: $G_t = R_{t+1} + \gamma G_{t+1}$.

$$V_t := V_t + \alpha(R_{t+1} + V_{t+1} - V_t)$$

Ці методи називають методами Монте-Карло. Їх недоліком є необхідність мати попередньо визначену траєкторію, щоб рахувати G_t починаючи з кінця траєкторії ($G_{T-1} = R_T, G_{T-2} = R_{T-1} + \gamma G_{T-1}$, ...).

Значно ефективнішим є Навчання тимчасовим різницям (TD). TD використовує замість сукупної винагороди G_{t+1}, її наближення V_{t+1}. Це дозволяє оновлювати функцію цінності на кожному кроці.

1.4. Методи наближеного розв'язання

До тепер, ми розглядали табличні методи, вони передбачають невеликі простори станів. Але в задачах, де стани це великі вектори, вони не є застосовними, тому замість них використовують апроксиматори: регресійні моделі, штучні нейронні мережі, чи навіть функціональні ряди.

Розглянемо перехід від табличних методів для нейронних мереж. Перш за все, вводять відхилення:

$$\overline{\mathrm{VE}}(\mathbf{w}) \doteq \sum_{s\in\mathcal{S}} \mu(s)\big[V(s) - \hat{V}(s,\mathbf{w})\big]^2,\ \mu - \text{розподіл станів}$$

Тоді градієнтний спуск виглядає так:

$$\begin{aligned}\mathbf{w}_{t+1} &\coloneqq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\big[V(S_t) - \hat{V}(S_t,\mathbf{w}_\mathrm{t})\big]^2\\ &= \mathbf{w}_t + \alpha\big[V(S_t) - \hat{V}(S_t,\mathbf{w}_\mathrm{t})\big]\nabla\hat{V}(S_t,\mathbf{w}_\mathrm{t})\\ &= \mathbf{w}_t + \alpha\big[U_t - \hat{V}(S_t,\mathbf{w}_\mathrm{t})\big]\nabla\hat{V}(S_t,\mathbf{w}_\mathrm{t})\end{aligned}$$

Де U_t (англ. unbiased) – незсунута оцінка $V(S_t)$ ($\mathbb{E}[U_t|S_t = s] = V(s)$). Наприклад, для методів Монте-Карло у ролі U_t виступає G_t – очікуване значення сукупної винагороди, що за означенням є незсунутим. Але для використання TD йдуть на поступки та використовують $\hat{G}_t = R_{t+1} + \gamma\hat{V}(S_{t+1},\ \mathbf{w_t})$, яка залежить від $\mathbf{w_t}$, що робить оцінку зсунутою, а методи напівградієнтними.

Одним з різновидів алгоритмів TD є Q-навчання. Це TD-алгоритм, де $\pi(a|S_{t+1})$ є жадібною стратегією – те саме, що $V(S_{t+1}) = \max\limits_a Q(S_{t+1},a)$.

Остаточно, градієнтний спуск для Q-навчання виглядає так:

$$\mathbf{w}_{t+1} \coloneqq \mathbf{w}_t + \alpha\left[R_{t+1} + \gamma\max_a \hat{Q}(S_{t+1},a,\ \mathbf{w_t}) - \hat{Q}(S_t,A_t,\mathbf{w}_\mathrm{t})\right]\nabla\hat{Q}(S_t,A_t,\mathbf{w}_\mathrm{t})$$

РОЗДІЛ 2. ДАНІ

2.1. Опис даних

Характеризуємо дані які використовуються перед кожним рішенням:

Перший вимір даних відповідає за моменти часу і має сталий крок. Біржі та аналітичні сервіси, що пропонують дані через API, зазвичай пропонують дані з кроком від одної хвилини до дня. Вочевидь, з ростом кроку збільшується випадкова складова. Тому, щоб мінімізувати невизначеність, слід використовувати найменший з доступних кроків (наприклад 1 хв).

Другий вимір даних містить показник. Ним часто нехтують, користуючись лише ціною. Тому з аналітичних сервісів на зразок Bybt були вивантажені 190 показників різних криптовалютних бірж: вхідні/вихідні об’єми, резерви, кількість переведень тощо.

З доступних до завантаження показників були вилучені такі, що є лінійними комбінаціями інших, а добутки та частки, навпаки були залишені. Такий відбір обумовлений природою нейронних мереж, які використовують лінійні комбінації ознак та не здатні апроксимувати добуток без достатньо великої кількості шарів.

Різні показники мають різні часові інтервали, тому вони були приведені до хвилинного шляхом заповнення наперед (forward fill). В результаті отримано набір даних (єдину таблицю) об’ємом у 10 Гб на 8 мільйонів рядків.

Таблиця 2 – Приклад заповнення наперед

datetime	reserve_usd_bitmex	reserve_usd_bitmex_ffill
2014-09-23 07:00:00+00:00	2.01281	2.01281
2014-09-23 07:01:00+00:00	nan	2.01281
…	…	…
2014-09-23 07:59:00+00:00	nan	2.01281
2014-09-23 08:00:00+00:00	1.16098	1.16098
2014-09-23 08:01:00+00:00	nan	1.16098
…	…	…

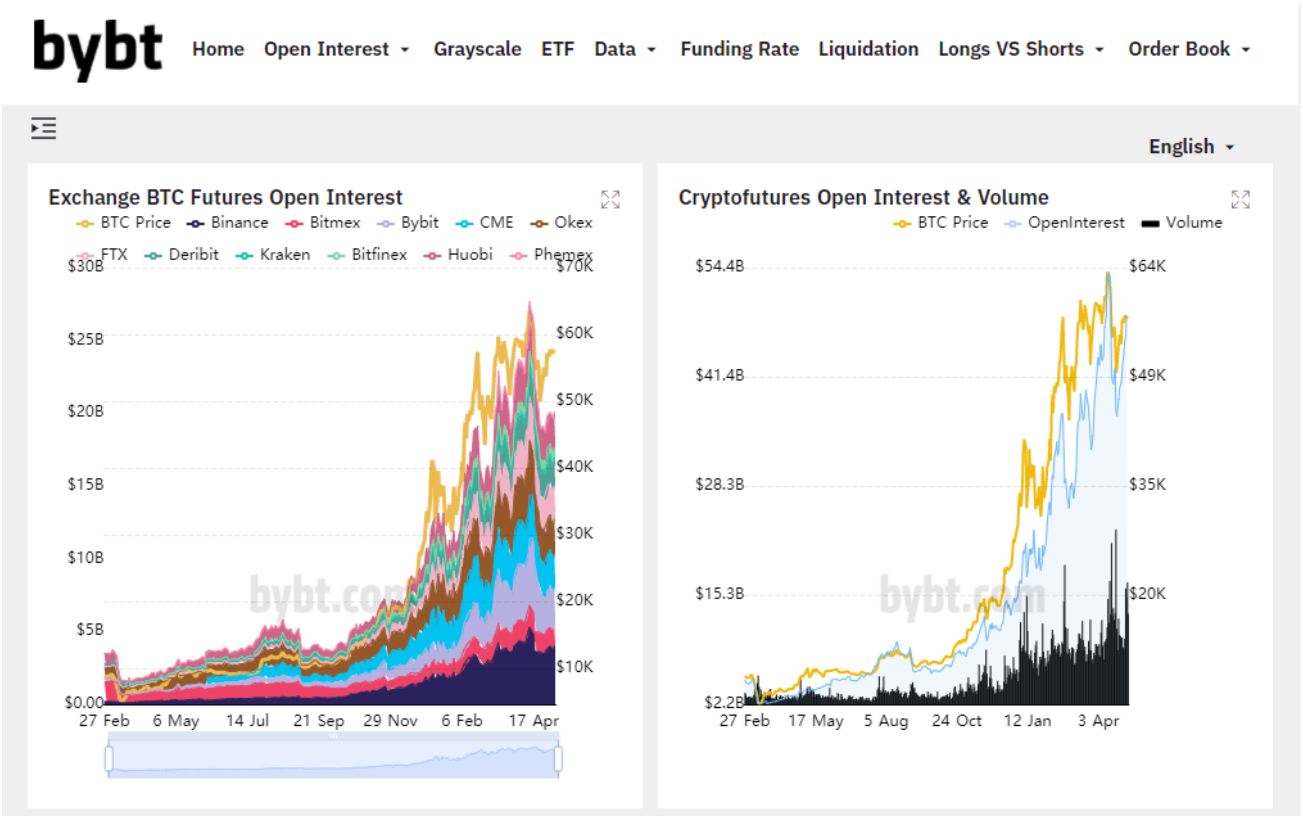

Рисунок 3 – Ф’ючерсна інформаційна платформа Bybt

Разом з показниками були вивантажені дані з біржі: ціна, дані для обчислення ціни ліквідації, роздільно об’єми покупок/продаж, які не надаються біржею і потребують локального обчислення із множини покупок/продаж.

2.2. Підготовка даних

2.2.1. Дробове диференціювання

Для унеможливлення зсуву розподілів реальних даних та даних тренування і валідації, як і в цілому для кращих передбачень, дані часові ряди мають бути стаціонарними. Щоб зробити ряд стаціонарним застосовують диференціювання. Похідну беруть відніманням від t’го елемента t-1’го. Недоліком цього методу є втрата корисної інформації, тому доцільним є використання дробового диференціювання [14]. Цей підхід використовує розкладання оператору лагу L ($Lx_t = x_{t-1}$) у біноміальний ряд (степеневий ряд із біноміальними коефіцієнтами), через що можливим стає диференціювання порядку $d \in (0,1)$:

$$\nabla^d x_t \stackrel{\text{def}}{=} (1-L)^d, \text{ тоді}$$

$$\nabla^d x_t = \sum_{k=0}^{\infty} \binom{d}{k} (-L)^k = 1 - dL - \frac{1}{2} d(1-d)L^2 + \cdots \approx$$

$$\approx \sum_{k=0}^{|x|} w_k x_{t-k}, \text{де } w_0 = 1, w_k = -w_{k-1} \frac{d-k+1}{k}$$

2.2.2. Згортки

Остання сума це згортка двох масивів. Якщо рахувати її як суму, то для отриманого набору даних це займатиме близько двох хвилин на колонку. Це неприйнятно багато, тому було використано згортку із розкладанням у ряд Фур'є [8] [15] (надалі це дозволить зменшити час обробки з двох місяців до семи годин).

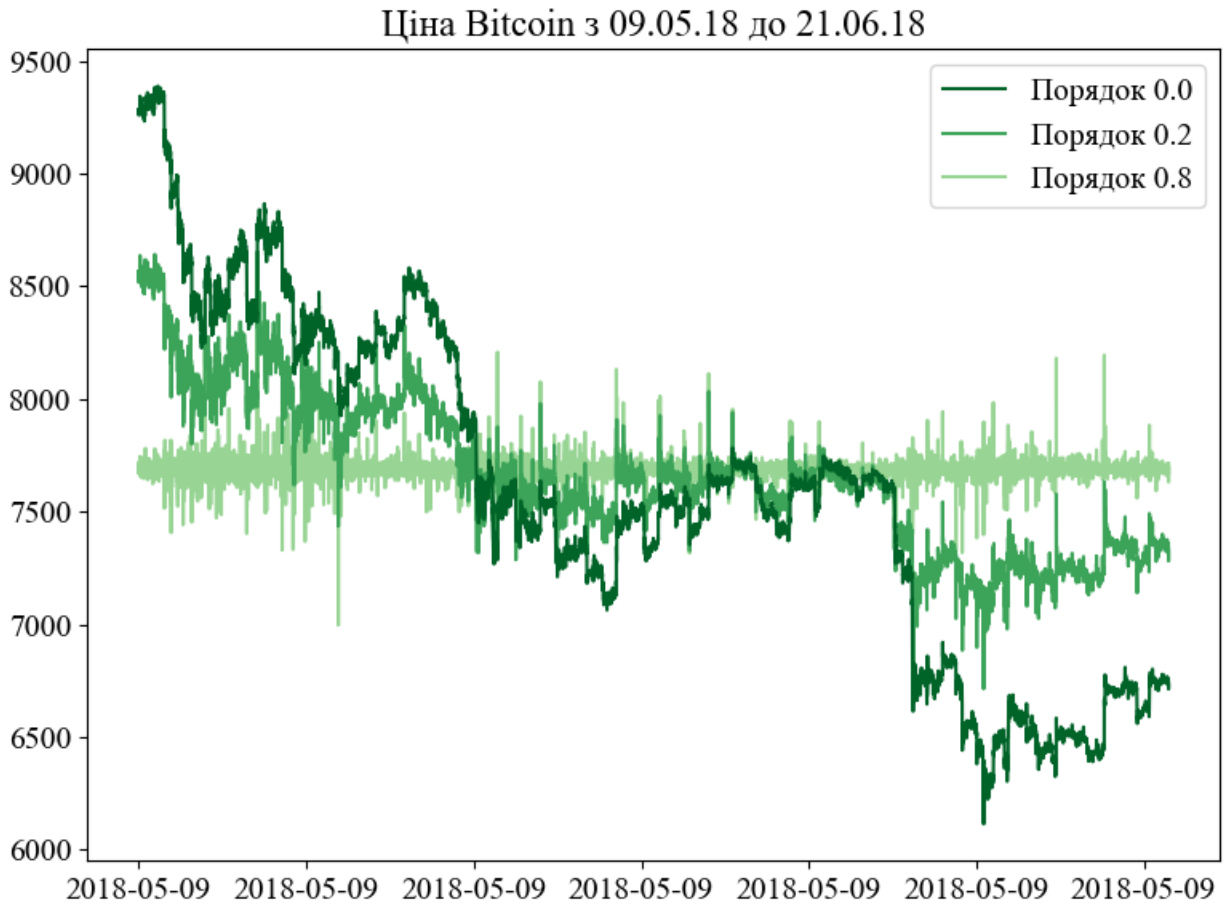

Рисунок 4 – Приклад дробового диференціювання

Згортка із FFT виглядає доволі просто:

```
ifft(fft(w) * fft(x))
```

2.2.3. Мінімальний порядок диференціювання

Чим менше буде порядок диференціювання, тим більше корисної інформації залишиться у даних. Щоб дізнатися який порядок призводить до стаціонарності, а який ні, можна скористатися методом поділу відрізка навпіл, а ще краще більш потужним методом Брента [16], для пошуку мінімального порядку.

Для перевірки на стаціонарність був використаний тест ADF-GLS [17]. Він є параметричним і страждає на хибно позитивні відповіді, тому його часто використовують у тандемі з KPSS [18], який є непараметричним методом, але страждає на хибно негативні відповіді. Тому однозначної відповіді про стаціонарність не може дати жоден з них.

На межі в яких був проведений пошук порядку диференціювання було накладене додаткове обмеження, яке не помічене в готових бібліотеках (fracdiff, numpy-fracdiff, fractional-differentiation-time-series). Фактично ми не можемо використовувати весь інтервал (0,1), через те що ми замінили нескінчений ряд який мав сходитись до 0, на скінчений ряд, що сходиться до додатного ε, яке зростає коли порядок наближається до 0 (Рис. 5). Через це може виникнути зайвий додатний зсув із невідомими наслідками. Тому слід визначити мінімальну межу для пошуку порядку. Це можна зробити введенням ε_{max}, наприклад за правилом трьох сигм:

```
e_max = 0.0027  # 1 - 3 sigma
min_order = scipy.optimize.brentq(
    lambda order:
        np.cumsum(get_weights(order, weights_num=len(df)))[-1] - e_max
    a=0, b=1,
)
```

Код 1 – Пошук мінімальної межі незсунутого порядку

за допомогою оптимізаційного методу Брента з бібліотеки SciPy [8]

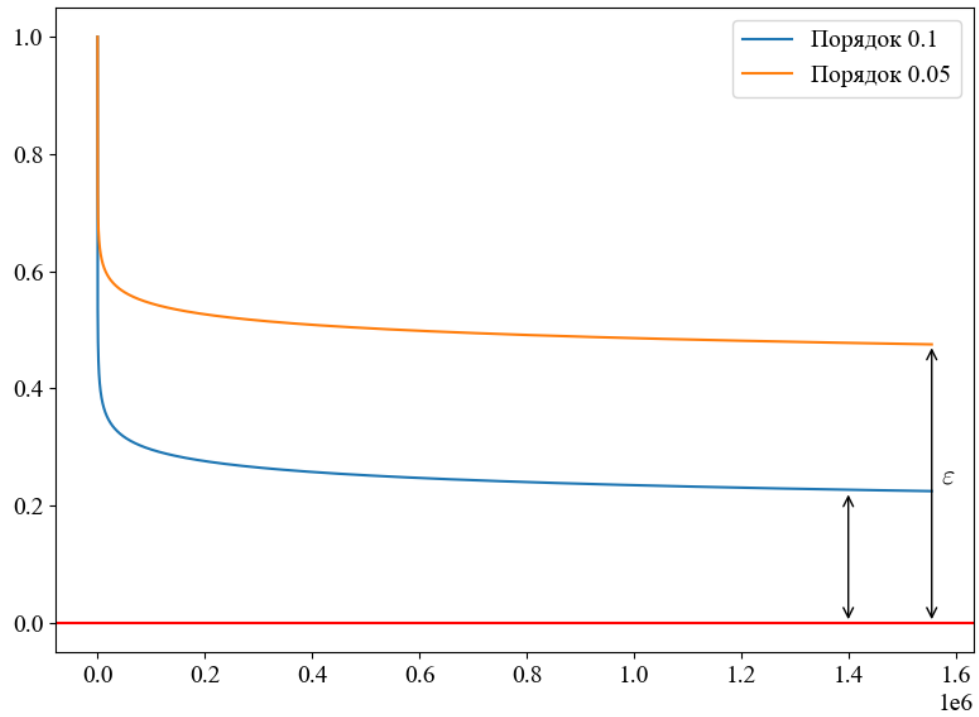

Рисунок 5 – Сукупна сума ваг дробового диференціювання

Під час тестів були використані моделі з константною моделлю регресії, бо зсув, який ми потім прибираємо під час стандартизації, присутній у всіх часових рядах. Для отримання зрівняних результатів пошук достатнього порядку диференціювання проведено зі сталою кількістю лагових компонент. Бо в економетричному пакеті arch [9], KPSS підбирає кількість лагів залежно від значень даних. Це може призвести до не виправдано великої кількості лагів. Тому вона вибирається як мінімальне ненульове значення при нульовому та одиничному порядках диференціювання.

```
def min_not_0(a, b):
    return 0 if a == b == 0
             else (max(a, b) if a == 0 or b == 0
                             else min(a, b))

lags_kpss = min_not_0(
    arch.KPSS(column, trend='c').lags,              # порядок 0
    arch.KPSS(numpy.diff(column), trend='c').lags  # порядок 1
)
diff_order_kpss = scipy.optimize.brentq(
    lambda order: arch.KPSS(
        lib.frac_diff_fft(column, order=order),
        trend='c', lags=lags_kpss
    ).pvalue - 0.0027,
    a=min_order, b=1
)
```

Код 2 – Пошук порядку диференціювання

Із-за оберненої альтернативної гіпотези та низького рівня прохідної значущості, слід вважати результат отриманий з KPSS необхідним, а з ADF-GLS – достатнім. Попри це, часто трапляються випадки, коли ADF-GLS стверджує про слабку стаціонарність ряду без жодного диференціювання, а KPSS потребує сильного диференціювання. Це може відбуватися через гетероскедастичність чи структурні розриви у даних. Приклад розбіжності у результатах тестів можна побачити у Табл. 3. Виключити ці колонки (116/141) ми не можемо через значні втрати обсягу інформації, не диференціювати теж. Тому доцільно диференціювати більшим з двох отриманих порядків.

Таблиця 3 – Результати тестування на відношення дисперсії, одиничний корінь та стаціонарність

Показник	Лаги KPSS	Порядок KPSS	Лаги DFGLS	Порядок DFGLS
transactions_count_outflow_bitmex	650	1	0	0
mpi	684	1	0	0
exchange_whale_ratio_spot_exchange	654	0.713	0	0
fund_flow_ratio_all_exchange	684	0.561	0	0
close	210	1	134	1
volume_buy	589	1	0	0
stablecoin_supply_ratio	685	0.815	0	1
thermo_cap	54	1	0	0.637
…				

Разом із тестами на одиничний корінь та на стаціонарність був проведений тест відношення дисперсії (variance-ratio test) [19]. Він перевіряє чи є часовий ряд випадковим блуканням, а його статистика дозволяє схарактеризувати одиничний корінь.

Таблиця 4 – Результати тесту відношення дисперсії

Показник	Відношення дисперсії	Значущість
close	1.024	0
volume_sell	0.643	0
reserve_usd_derivative_exchange	0.999	0.472129
reserve_usd_bitmex	1	0.785474
inflow_top10_all_exchange	1	0
…		

Статистика близька до 1 відповідають типовому випадку одиничного кореня. Статистика між 0 та 1 показує, що після збурення, значення величини знаходиться між початковим та піковим значеннями. Статистика більше 1 говорить про те, що після збурення значення величини стає ще більшим.

Згідно з цим тестом більшість показників (137/141) мають типовий одиничний корінь, також більшість не є випадковими блуканнями (121/141).

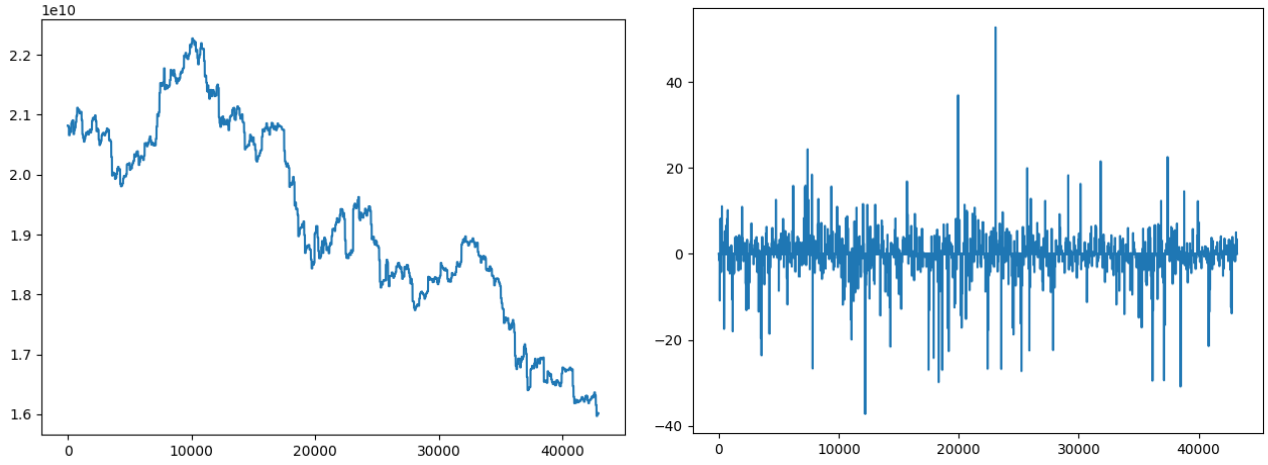

Рисунок 6 – Дані сукупного резерву бірж до/після передоброблення. Лівий часовий ряд нестаціонарний та має тренд, правий є стаціонарним

Після стаціонаризації, часові ряди були стандартизовані, для швидшого навчання і менших ваг нейронів.

РОЗДІЛ 3. СЕРЕДОВИЩЕ ТА АГЕНТ

3.1. Архітектура нейронної мережі

На відміну від задач комп’ютерного зору, де кожний піксель поодинці, не несе корисної інформації, ми маємо дані, в яких кожний показник може потенційно бути визначним. Також відомо, що широкі нейронні мережі мають гарні здібності у запам’ятовуванні. Тому слід мати достатньо широку нейронну мережу, наприклад, шириною у кількість входів (або пропорційно).

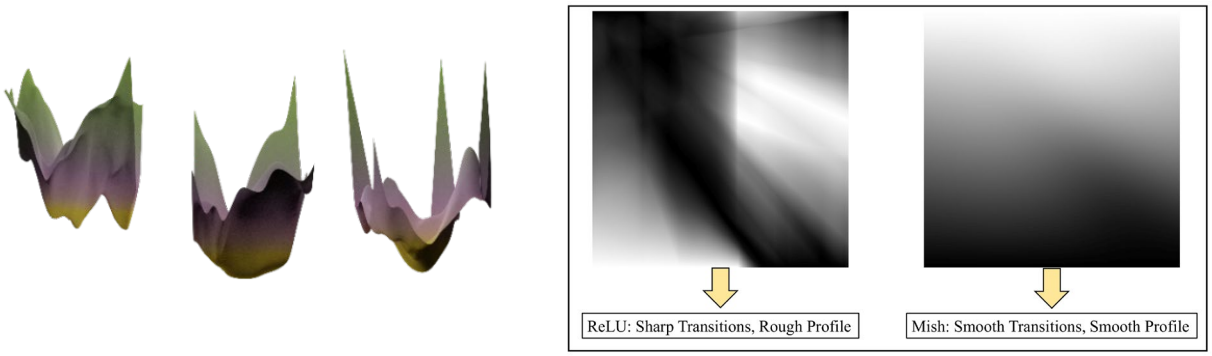

Рисунок 7 – Порівняння вихідних поверхонь [20]
1: ReLU/Swish/Mish, 2: ReLU/Mish

У якості функції активації, була використана Mish [20]. Вона дає більш гладку поверхню функції втрат. Для регуляризації був використаний ConcreteDropout [21], що має адаптивний показник вимикання нейронів.

Для ініціалізації шарів із Mish був використаний рівномірний розподіл Xi [22], для останнього шару із лінійною активацією – рівномірний розподіл Глорота [23]. Встановлено, що ці розподіли запобігають розмиванню градієнта [22] [23].

3.2. Реалізація середовища

Існує багато готових середовищ: у складі бібліотеки Tensortrade, у складі бібліотеки FinRL, gym-anytrading, TradingGym, trading-gym. Але жодне з них не є одночасно: і перевіреним на крайові випадки, і достатньо швидким, і таким, що включає комісію та враховує відхилення ціни закриття від можливої ціни виконання. Тому було прийнято рішення розробити власне середовище на базі OpenAI Gym.

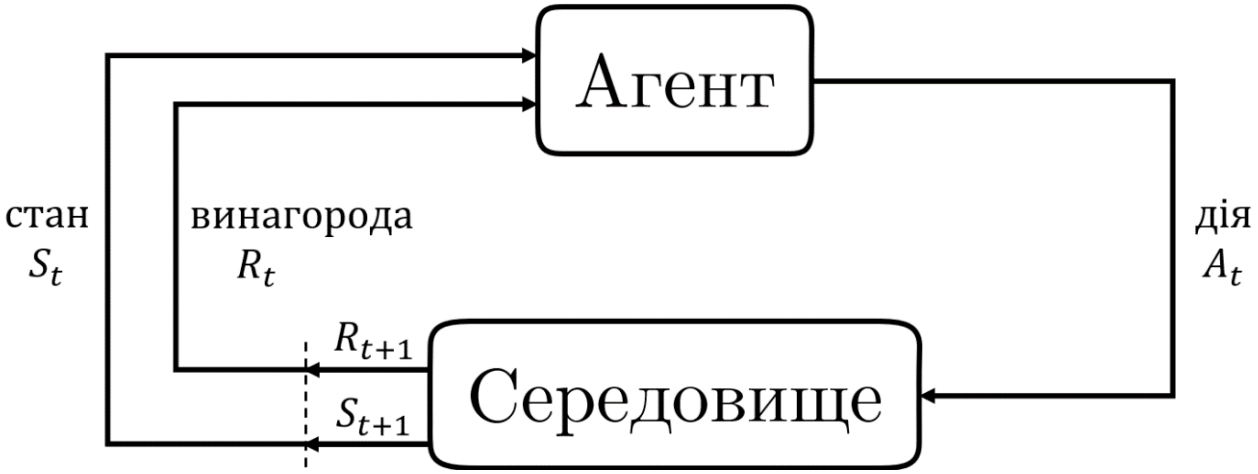

Рисунок 8 – Типова схема взаємодії агента з середовищем

Створене середовище повністю копіює поведінку біржі Bitmex, всі заокруглення та комісії:

```
def value_diffs(self) -> np.ndarray:
    if self.buys:
        enter_values = np.fromiter(
            (lib.contract_to_xbt(
                # поправка покупки:
                price=buy_price + self.corr_buy,
                multiplier=-self.instrument.multiplier
            ) for buy_price in self.buys),
            # заокруглення:
            dtype=np.int,
        )

        one_buy_exit_value = lib.contract_to_xbt(
            # поправка продажі:
            price=self.close[self.t] - self.corr_sell,
            multiplier=-self.instrument.multiplier,
        )

        # комісії:
        return (
            (1 - self.position.commission) * enter_values -
            (1 + self.position.commission) * one_buy_exit_value
        )
    else:
        return np.array([])
```

Код 3 – Обчислення різниць об’ємів покупок і продаж

Також воно є високопродуктивним через:

1) використання структурованих масивів numpy.recarray, замість типових для аналізу даних, таблиць pandas.DataFrame, що уповільнюють обчислення через велику кількість Python-абстракцій
2) використання лінивих генераторів
3) використання view замість копій масивів
4) кешування, наприклад під час обрахунку кількості сатоші (1e-8 біткоїну) в одному доларі:

```
@lru_cache(maxsize=None)
def contract_to_xbt(price, multiplier) -> int:
    return round(multiplier / price)
```

3.2.1. Спостереження

На кожному кроці середовище повертає дані показників разом із кортежем, що характеризує стан позиції: (*value_diffs_mean*, *buys_age* / *max_buys_age*, *buys_age* > *max_buys_age*). Перше значення рахуємо як середнє поелементного віднімання об'ємів покупок від об'єму, що буде вивільнено, якщо продаж відбудеться по поточній ціні. Друге і третє значення показують агенту скільки він ще може знаходитися у позиції, бо інакше це буде грою із нескінченими сумами.

Дуже важливим є те, як стандартизувати стан позиції. Бо якщо привести його до неправильного масштабу, то на початку нейронна мережа буде сприймати його більш або менш важливими за дані показників. Останні два значення дуже просто привести до відрізка [-1, 1], бо відомі їх межі:

$$((buys\_age / max\_buys\_age) - 0.5) * 2$$

$$1 \text{ if } buys\_age > max\_buys\_age \text{ else } -1$$

Зсув і масштаб першого значення рахуємо як середнє і стандартне відхилення різниць у межах вікна, що рухається у часі та охоплює часовий проміжок розміром у максимальний вік позиції. Наприклад, якщо $max\_buys\_age = 2$, то в кожному вікні буде по дві різниці:

values = [1, 2, 3, 4, 5]

value_windows = [[1, 2, 3], [2, 3, 4], [3, 4, 5]]

window_diffs = [[3 - 1, 3 - 2], [4 - 2, 4 - 3], [5 - 3, 5 - 4]] =

= [[2, 1], [2, 1], [2, 1]]

powers_means = mean(*value_diffs*) = [2, 1]

value_diff_mean = mean(*powers*) = 1.5

value_diff_std = std(*diff_power_means* - *value_diff_mean*) =

= std([2 - 1.5, 1 - 1.5]) = std([0.5, -0.5]) = 0.5

3.2.2. Функція винагороди

Використовувалася найпростіша винагорода: різниця об’ємів покупки/продажі.

3.3. Агент

Існує багато видів агентів основний перелік разом з описом наведено у таблиці:

Таблиця 5 – Основні агенти. D/C – дискретний/неперервний простори

Алгоритм	Опис	Стратегія	Простір дій	Простір станів	Оператор
Q-learning	State–action–reward–state	Поза	D	D	Q-значення
SARSA	State–action–reward–state–action	Вздовж	D	D	Q-значення
DQN	Deep Q Network	Поза	D	C	Q-значення
DDPG	Deep Deterministic Policy Gradient	Поза	C	C	Q-значення
A2(3)C	(Asynchronous) Advantage Actor-Critic	Вздовж	C	C	Функція переваги

NAF	Q-Learning with Normalized Advantage Functions	Поза	C	C	Функція переваги
TRPO	Trust Region Policy Optimization	Вздовж	C	C	Функція переваги
PPO	Proximal Policy Optimization	Вздовж	C	C	Функція переваги
TD3	Twin Delayed DDPG	Поза	C	C	Q-значення
SAC	Soft Actor-Critic	Поза	C	C	Функція переваги

У цій роботі були проведені експерименти із DQN [2] [3]. Це Q-навчання покращене використанням буфера повторів, що після кожного кроку зберігає кортеж переходу (S, A, R, S'). Навчання з буфером повторів ведеться не з останнього отриманого кортежу, а з набору випадкових, що містяться у буфері. Це сприяє більшому узагальненню та робить алгоритм ефективнішим, питомо до кількості спостережень.

3.4. Обмеження гіперпараметрів

ε-жадібна стратегія з імовірністю ε закриває позицію, що рівносильно запуску середовища не в нульовий, а у випадковий момент часу. Це має добре вплинути на здібності до узагальнення, проте, середня відстань між стартами має бути не меншою за максимальний вік покупки. Тому (3 є розміром простору станів {BUY, SELL, HOLD}):

$$\frac{\varepsilon}{3} max\_steps < max\_buy\_age$$

РОЗДІЛ 4. РЕЗУЛЬТАТИ ТРЕНУВАННЯ

4.1. Результати без зсуву та комісії

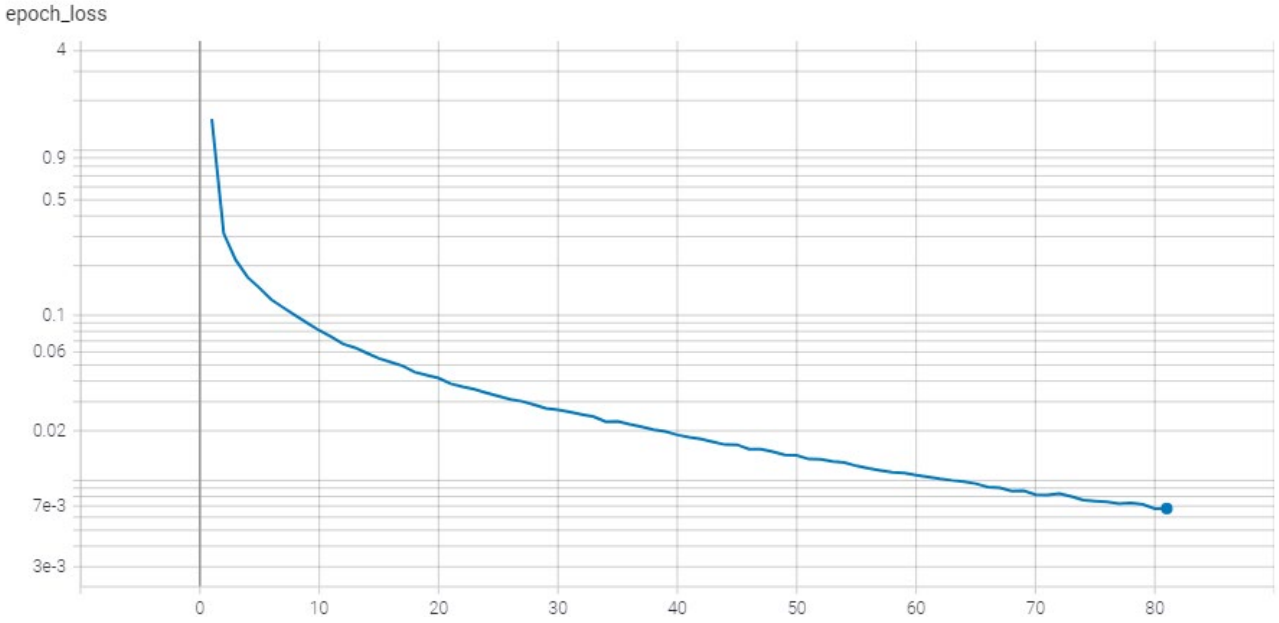

Рисунок 9 – Функція відхилення (логарифмічна шкала)

Надалі, під доходом маємо на увазі 1 отриманий долар до 1 вкладеного, по курсу 10'000$/BTC.

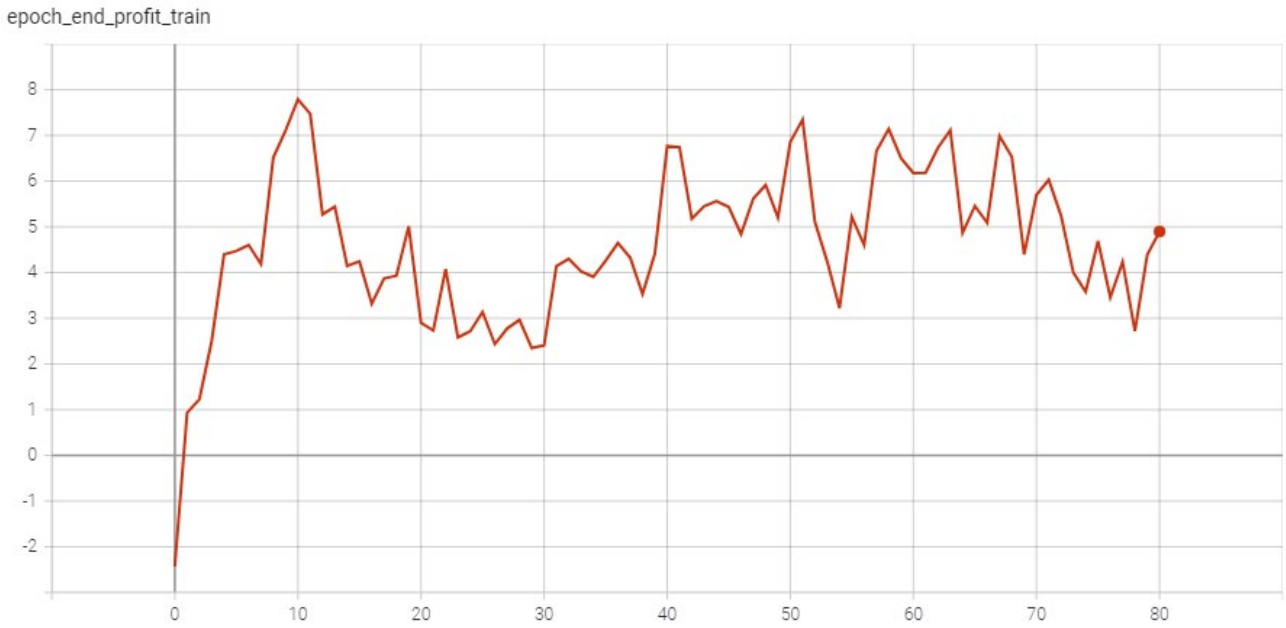

Рисунок 10 – Дохід епохи під час тренування

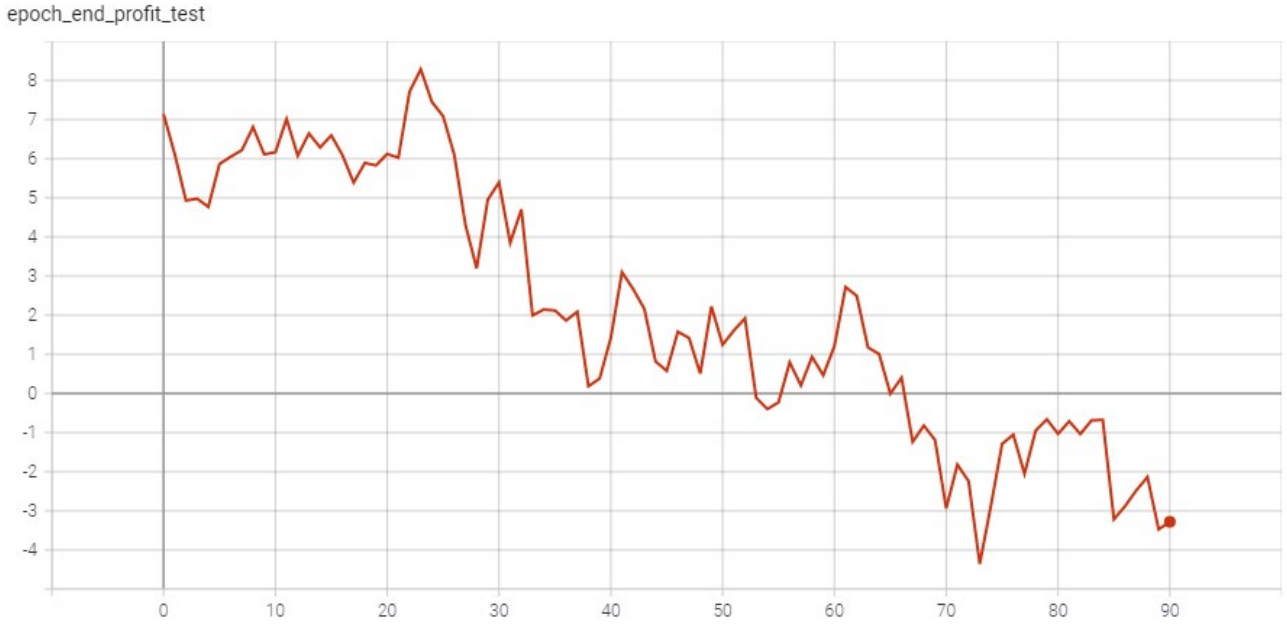

Рисунок 11 – Дохід епохи під час тестування

4.2. Результати зі зсувом та комісією

Після тренування без зсуву та комісії найкращі ваги були використані для оптимістичної ініціалізації агента у середовищі зі зсувом та комісією.

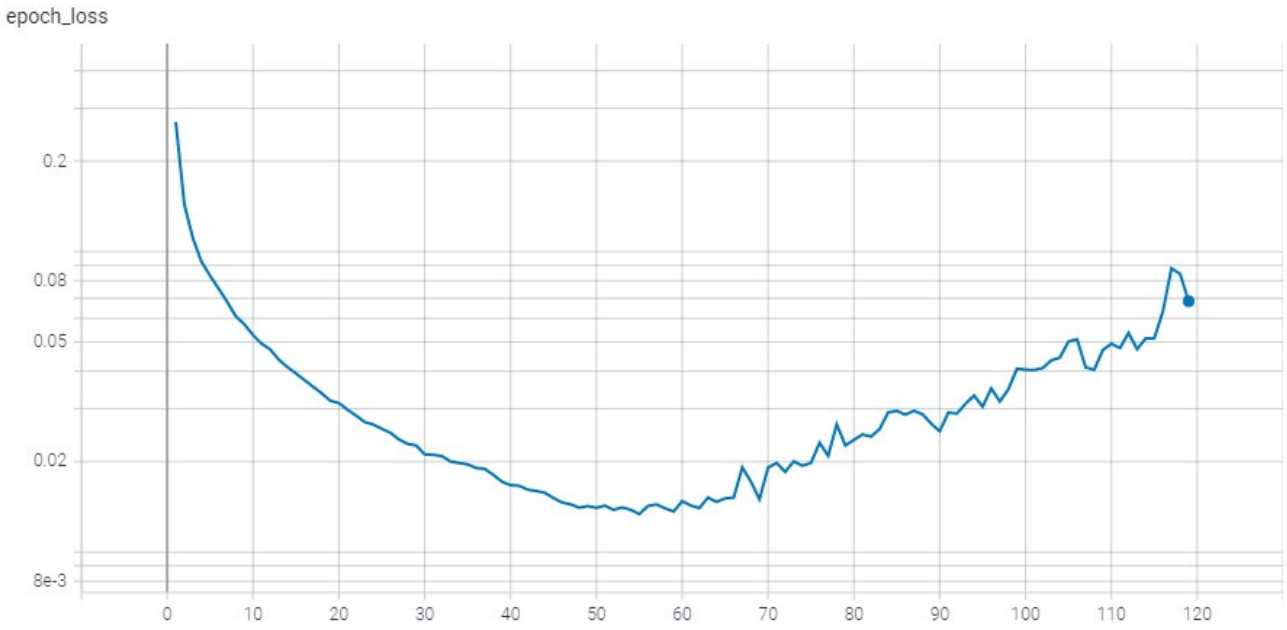

Рисунок 12 – Функція відхилення (логарифмічна шкала)

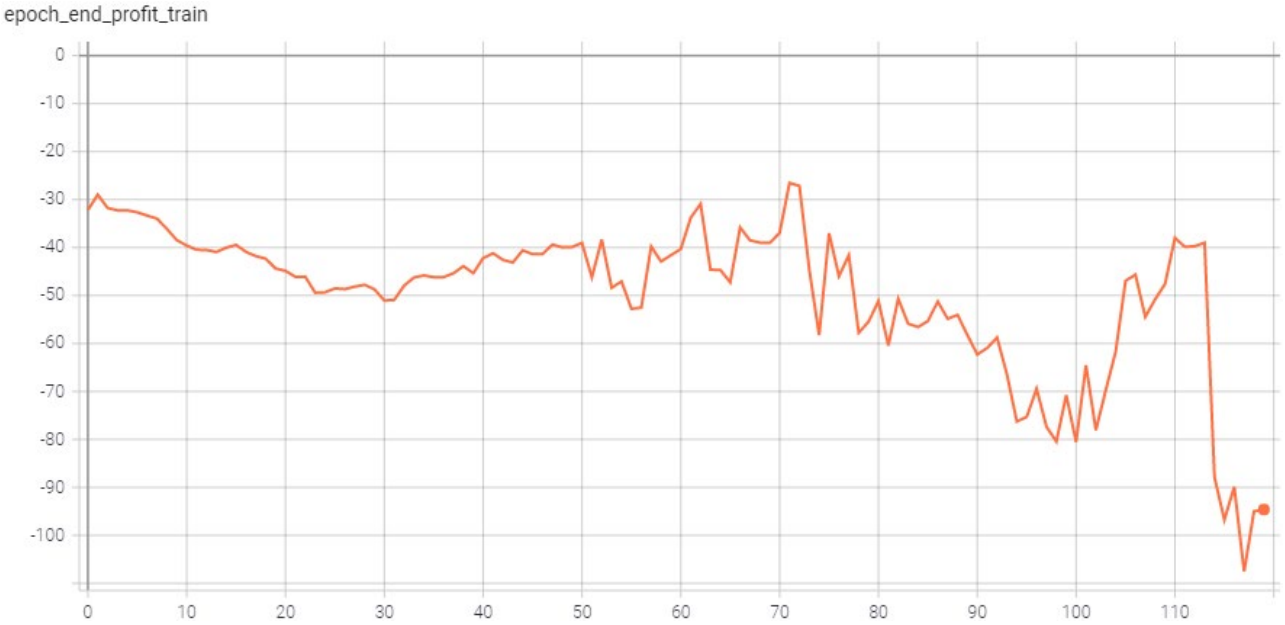

Рисунок 13 – Дохід епохи під час тренування (оптимістична ініціалізація)

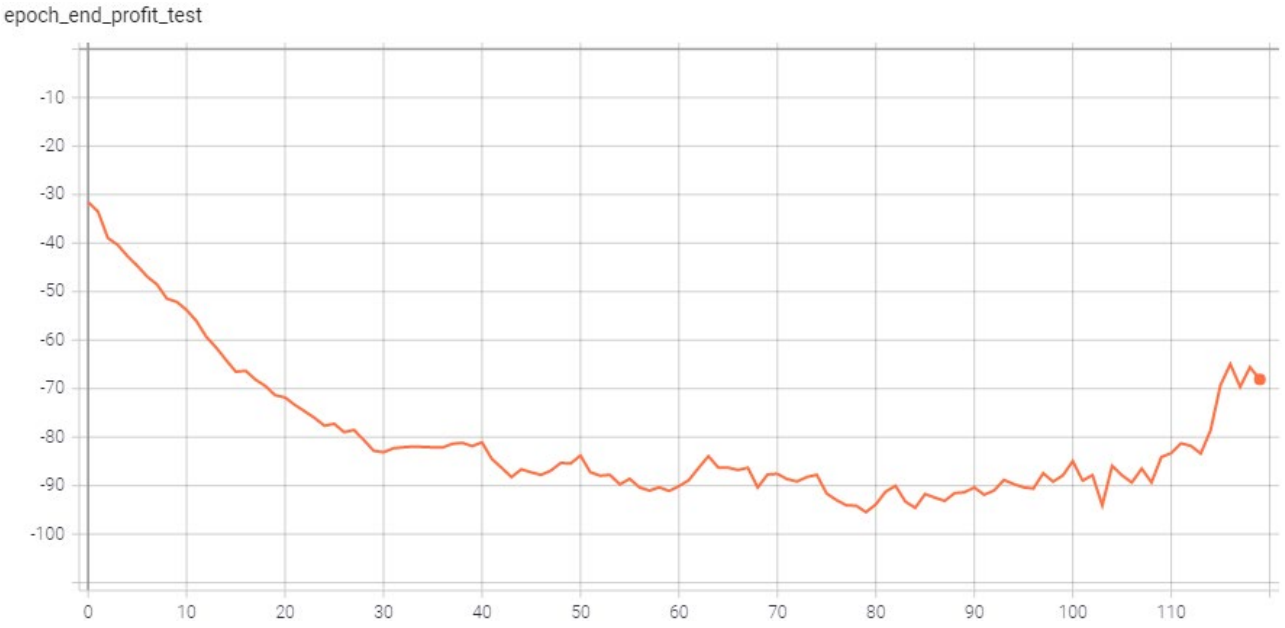

Рисунок 14 – Дохід епохи під час тестування (оптимістична ініціалізація)

ВИСНОВКИ

Через відсутність функції відхилення під час тестування (валідації), ми можемо робити висновки лише з доходу. Результати без зсуву і комісії складно інтерпретувати, бо до кінця не зрозуміло, чи є перенавчання причиною падіння доходу. Проте стабільний високий прибуток упродовж перших двадцяти епох свідчить про працездатність в ідеалізованому середовищі.

Значно гірший результат маємо зі зсувом та комісією: дохід є від'ємним без жодного натяку на зростання.

Можливо слід спробувати застосувати покращену версію DQN – Rainbow [24]. Найбільшими її перевагами є використання пріоритетного буфера повторів [25] та багатокрокових винагород при оновленні функції цінності, що може допомогти в прийнятті більш продуманих рішень. Також доцільною є заміна ε-жадібної стратегії на верхню довірчу границю [26], це дозволить зважати на те, коли останній раз виконувалася випадкова дія, для поступового, а не хаотичного розвідування.

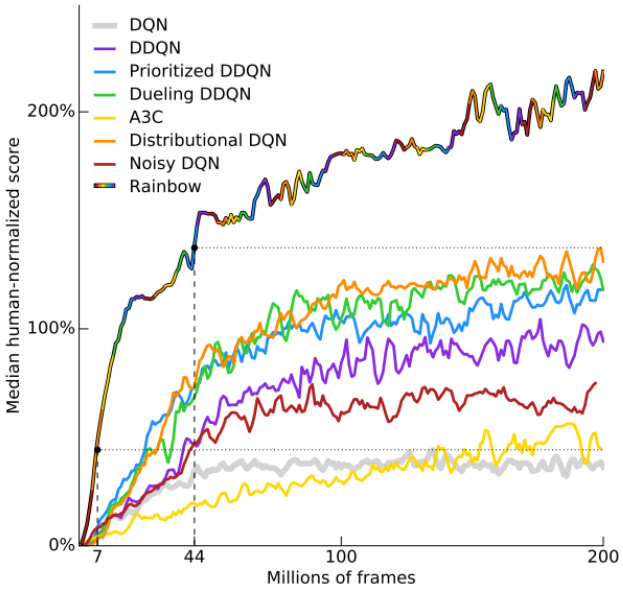

Рисунок 15 – Порівняння продуктивності різних версій DQN [24] у навчальному середовищі аркадних ігор Atari 2600 [27]

Більш радикальною була б відмова від навчання з підкріпленням на користь еволюційних алгоритмів: генетичного програмування, генетичних алгоритмів чи нейроеволюції [28].

СПИСОК ЛІТЕРАТУРИ

1. Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., та Zaremba W. OpenAI Gym. 2016.
2. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., та Riedmiller M.A., "Playing Atari with Deep Reinforcement Learning," // ArXiv, № abs/1312.5602, 2013.
3. Mnih V., Kavukcuoglu K., Silver D., Rusu A.A., Veness J., Bellemare M.G., Graves A., Riedmiller M.A., Fidjeland A., Ostrovski G., та ін., "Human-level control through deep reinforcement learning," // Nature, № 518, 2015. С. 529-533.
4. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., та ін. TensorFlow: A system for large-scale machine learning // OSDI. 2016.
5. Chollet F., others. Keras. 2015.
6. Plappert M. keras-rl. GitHub, 2016.
7. Harris C.R., Millman K.J., van der Walt S.J., Gommers R., Virtanen P., Cournapeau D., Wieser E., Taylor J., Berg S., Smith N.J., та ін., "Array programming with NumPy," // Nature, № 585, Вересень 2020. С. 357–362.
8. Virtanen P., Gommers R., Oliphant T.E., Haberland M., Reddy T., Cournapeau D., Burovski E., Peterson P., Weckesser W., Bright J., та ін., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," // Nature Methods, № 17, 2020. С. 261–272.
9. Sheppard K., Khrapov S., Lipták G., mikedeltalima, Capellini R., Hugle, esvhd, Fortin A., JPN, Li W., та et al., "bashtage/arch: Release 4.19," 2021.
10. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., та ін., "Scikit-learn: Machine Learning in Python," // Journal of Machine Learning Research, № 12, 2011. С. 2825–2830.

11. pandas development team T. pandas-dev/pandas: Pandas. Zenodo, 2020.
12. Hunter J.D., "Matplotlib: A 2D graphics environment," // Computing in Science & Engineering, № 9, 2007. С. 90–95.
13. Sutton R., Barto A., "Reinforcement Learning: An Introduction," // IEEE Transactions on Neural Networks, № 16, 2005. С. 285-286.
14. Hosking J.R.M., "Fractional differencing," // Biometrika, № 68, 1981. С. 165-176.
15. Jensen A.N., Nielsen M., "A Fast Fractional Difference Algorithm," // Econometrics: Mathematical Methods & Programming eJournal, 2013.
16. Brent R., "Table errata: Algorithms for minimization without derivatives (Prentice-Hall, Englewood Cliffs, N. J., 1973)," // Mathematics of Computation, № 29, 1975. P. 1166.
17. Elliott G., Rothenberg T., та Stock J., "Efficient Tests for an Autoregressive Unit Root," // Econometrica, № 64, 1996. С. 813-836.
18. Kwiatkowski D., Phillips P., Schmidt P., та Shin Y., "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?," // Journal of Econometrics, № 54, 1992. С. 159-178.
19. Campbell J., Lo A., Mackinlay A., та Whitelaw R.F. The Econometrics of Financial Markets. Princeton University Press, 1996. С. 48–55.
20. Misra D., "Mish: A Self Regularized Non-Monotonic Neural Activation Function," // ArXiv, № abs/1908.08681, 2019.
21. Gal Y., Hron J., та Kendall A. Concrete Dropout // NIPS. 2017.
22. He K., Zhang X., Ren S., та Sun J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," // 2015 IEEE International Conference on Computer Vision (ICCV), 2015. С. 1026-1034.
23. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks // AISTATS. 2010.

24. Hessel M., Modayil J., Hasselt H.V., Schaul T., Ostrovski G., Dabney W., Horgan D., Piot B., Azar M.G., та Silver D. Rainbow: Combining Improvements in Deep Reinforcement Learning // AAAI. 2018.
25. Schaul T., Quan J., Antonoglou I., та Silver D., "Prioritized Experience Replay," // CoRR, № abs/1511.05952, 2016.
26. Auer P., "Using Confidence Bounds for Exploitation-Exploration Trade-offs," // J. Mach. Learn. Res., № 3, 2002. С. 397-422.
27. Bellemare M.G., Naddaf Y., Veness J., та Bowling M. The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract) // IJCAI. 2015.
28. Such F.P., Madhavan V., Conti E., Lehman J., Stanley K., та Clune J., "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," // ArXiv, № abs/1712.06567, 2017.

ДОДАТОК А.
РЕАЛІЗАЦІЯ ДРОБОВОГО ДИФЕРЕНЦІЮВАННЯ

```
from typing import Optional

import numpy as np
import scipy.signal
import scipy.special
import scipy.stats
from numba import jit


def get_weights(order: float, weights_num: int, mode: int = 0) -> np.ndarray:
    if mode == 0:
        k = np.arange(1, weights_num)
        weights = np.concatenate(([1], np.cumprod((k - order - 1) / k)))
    elif mode == 1:
        s = np.tile([1.0, -1.0], -(-weights_num // 2))[:weights_num]
        weights = s * scipy.special.binom(order, np.arange(weights_num))
    else:
        raise mode

    return weights


def get_weights_eps(order: float, max_weights_num: int, eps: float = EPS) -> np.ndarray:
    weights = [1.]
    for k in range(max_weights_num - 1):
        w = weights[-1] * -(order - k) / (k + 1)
        if abs(w) <= eps:  # <= 0
            break
        weights.append(w)
    return np.array(weights)


def num_from_threshold(weights, min_rel_imp):
    weight_sums = np.cumsum(np.abs(weights[::-1]))
    weight_sums = weight_sums / weight_sums[-1]
    take_num = len(weight_sums[(weight_sums >= min_rel_imp) & (weight_sums > 0)])
    return take_num


def diff_numba(array: np.ndarray, weights: np.ndarray):
    res = np.zeros_like(array, dtype=np.float64)
    shift = np.zeros(array.size * 2)
    shift[array.size:] = array
    for i, w in enumerate(weights):
        res += w * shift[array.size - i: 2 * array.size - i]  # ~ w * shift_right(array, i)
    return res
```

```
def frac_diff(array: np.ndarray, order: float, max_weights_num: Optional[int] = None,
              min_rel_imp: Optional[float] = None, all_weights=False, do_skip=True,
              numba_kwargs: Optional[dict] = None):
    assert isinstance(array, np.ndarray)
    assert len(array.shape) == 1, array.shape
    assert 0 <= order <= 1, order

    if numba_kwargs is None:
        numba_kwargs = {'parallel': True, 'nopython': True}

    if min_rel_imp is not None:
        assert 0 <= min_rel_imp <= 1, min_rel_imp
        weights = get_weights_eps(order, max_weights_num=array.size)
        if all_weights:
            skip = num_from_threshold(weights, min_rel_imp) - 1
        else:
            take_weights = num_from_threshold(weights, min_rel_imp)
            weights = weights[:take_weights]
            skip = take_weights - 1
    elif max_weights_num is not None:
        assert 1 <= max_weights_num <= array.size, (1, max_weights_num, array.size)
        weights = get_weights_eps(order, max_weights_num)
        skip = len(weights) - 1
    else:
        raise (max_weights_num, min_rel_imp)

    if skip / array.size > 0.1:
        print('Skip ratio', skip, array.size, skip / array.size)

    res = jit(**numba_kwargs)(diff_numba)(array, weights)

    return res[skip if do_skip else 0:]
```

```python
def frac_diff_fft(array: np.ndarray, order: float, skip_threshold: Optional[float] = None, do_skip=True):
    assert isinstance(array, np.ndarray)
    assert len(array.shape) == 1, array.shape
    assert 0 <= order <= 1, order

    if skip_threshold is None:
        skip_threshold = n_sigma_p(n=3)
    weights = get_weights(order, weights_num=array.size)
    if order == 0:
        return array
    elif order == 1:
        return np.diff(array)
    else:
        c1 = np.cumsum(weights)
        c1 = c1 / c1.max()

        c2 = np.cumsum(c1)
        c2 = c2 / c2.max()

        c3 = np.diff(c2)
        c3 = c3 / c3.max()

        skip = len(c3[c3 > n_sigma_p(skip_threshold)])

        if skip > array.size * 0.2:
            print('Size', order, skip, array.size, array.size * 0.2)
            print('Consider decrease threshold')

    res = scipy.signal.convolve(array, weights, mode='full')[:array.size]
    # ~res = scipy.fft.irfftn(scipy.fft.rfftn(array) * scipy.fft.rfftn(weights))

    return res[skip if do_skip else 0:]
```