Санкт–Петербургский государственный университет
Кафедра математической теории игр и статистических решений

Кудин Павел Сергеевич

Выпускная квалификационная работа

Применение алгоритмов обучения с подкреплением для управления системами электроснабжения с возобновляемыми источниками энергии

Направление 01.04.02
«Прикладная математика и информатика»
Магистерская программа:
Исследование операций и системный анализ

Научный руководитель:
кандидат физико-математических наук,
доцент кафедры ММЭС,
Петросян Ованес Леонович

Рецензент:
руководитель группы общества с ограниченной
ответственностью «Техкомпания Хуавей»,
Михеев Викентий Сергеевич

Санкт-Петербург
2022 г.

Содержание

Введение

Системы накопления энергии и возобновляемые источники энергии занимают важное место в оптимальном планировании работы энергосистем. Инжиниринговой компанией Schneider Electric [1], специализирующейся на энергетическом менеджменте и автоматизации, были опубликованы данные в рамках соревнования Power Laws [2]. Главной целью стала разработка оптимизационной модели для минимизации финансовых затрат на электроэнергию за счет планирования зарядки и разрядки аккумуляторной батареи, а также обмена с энергетическим рынком при условиях соблюдения ограничений системы и достижения энергетического баланса.

В рамках соревнования участниками были предложены решения [2], основывающиеся на детерминистических подходах и не позволяющие применять их в условиях реальных индустриальных задач. Однако, в области планирования в энергосистемах стали популяризироваться подходы, основанные на обучении с подкреплением (Reinforcemenet Learning (RL))[3]-[5], в связи с их возможностью к адаптации в среде задачи в случае изменения входных параметров или столкновения с непредвиденными ранее ситуациями, а также способностью к работе с множеством неопределенностей, содержащихся в исторических данных.

В данной работе автором проводится анализ недостатков в применении смешанного целочисленного линейного программирования (MILP) при решении исследуемой задачи и предлагается альтернативный подход, основанный на использовании методов RL, алгоритмы которого в последнее время становятся более популярными в сфере использования накопительных систем в энергосистемах. На основе имеющихся исторических данных, проводится обучение трех моделей RL категории on-policy learning с помощью Ray API [6], включая описание процесса определения среды рассматриваемой задачи, использование параллелизма в обучении модели, а также результаты и визуализацию полученных результатов в сравнении с MILP.

Постановка задачи

Объект исследования – система электроснабжения, состоящая из фотоэлектрической электростанции генерирующей солнечную энергию, накопителя энергии в виде аккумуляторной батареи и обладающей возможностью обмена энергией (купли / продажи) с местной коммунальной сетью по заранее известным тарифам.

Цель работы – получение модели планирования подхода RL, способной в режиме реального времени оптимально использовать аккумуляторную батарею для минимизации финансовых затрат на покупку энергии у местной коммунальной сети с необходимым условием в удовлетворении спроса на энергию в энергосистеме.

В рамках данной работы были решены следующие задачи:

- проведена математическая постановка задачи в терминах линейного программирования
- выполнен обзор имеющихся в задаче данных
- выявлены недостатки детерминистического подхода в рамках рассматриваемой задачи и необходимость перехода к обучению с подкреплением
- сделано описание основных принципов, понятий и методов RL
- приведен обзор алгоритмов RL использованных при решении задачи
- определена и разработана среда в рамках задачи RL на основе имеющихся данных
- выполнено проведение экспериментов и сравнение результатов работы полученных моделей RL с результатами MILP

Обзор литературы

Исследования в области использования накопительных систем в энергосистемах чаще всего подразделяются на 2 основных направления. Первое берет во внимание снижение финансовых затрат или внешнего воздействия на среду путем управления батареями [7] – [10]. Второе касается изучения проблемы расчета оптимального размера накопителей для повышения эффективности их дальнейшего использования [11, 12]. Некоторые научные работы акцентируют свое внимание на проблемах оптимального управления батареей, рассматривая задачи стабилизация энергии при зарядке и разрядке батареи [13] и уменьшение эффекта ее деградации [14].

В последнее время в области задач связанных с энергосистемами начинают активно применяться подходы, основанные на использовании обучения с подкреплением. Например, в работе [15] авторы использовали RL для получения прогнозного плана на основе жилой нагрузки, а в [16] была разработана модель управления питанием в режиме реального времени в рамках гибридной системы хранения энергии с использованием электромобиля. В исследовании [17] для энергосистемы был предложен алгоритм динамического ценообразования и планирования потребления энергии с использованием подходов RL.

Обучение с подкреплением – одна из областей машинного обучения, в основой для которой является процесс взаимодействий агента и среды, в результате которых агентом получается некоторая информация о состоянии среды посредству выбора действий. Довольно быстро популярностью стали пользоваться алгоритмы глубокого обучения с подкреплением (Deep RL), позволяющие объединять традиционные методы RL, дополняя их использованием нейронных сетей для решения задач, в которых невозможно ограничиться только лишь классическими подходами. В основе любого алгоритма обучения с подкреплением для его описания используется марковский процесс принятия решений [18].

Работая с методами RL особое внимание необходимо уделять рассматриваемой проблеме, а именно наличию непрерывности ее пространств состояний и действий. В рамках исследуемой задачи алгоритмам необхо-

димо извлечь наиболее весомые характеристики из исторических данных, построив модель оптимального управления. Дискретизация пространства действий или состояний влияет на скорость и процесс сходимости рассматриваемых методов. Имеются примеры работ, в которых за счет использования методов Q-learning и Deep Q-learning удалось продемонстирровать хорошие результаты в рамках задач связанных с управлением ветряной станцией [19], формированием энергитических торговых стратегий [20] и некоторых проблем с наличием неопределенностей [21, 22].

Одним из наиболее примечательных классических алгоритмов при решении задач методом обучения с подкреплением является Policy Gradient (PG) [23]. В его основе лежит использование градиентного подъема при в итерационном процессе обновления параметров стратегии агента. Данный метод чаще всего является лишь основой для более модернизированных с точки зрения используемых техник алгоритмов к решению различных задач.

Proximal Policy Optimization (PPO) [24] наследует идею алгоритма Policy Gradient, изменяя подход к обновлению параметров стратегии агента. Вместо необходимости в вычислении производных второго порядка, предлагается перейти к использованию градиентного спуска с добавлением в целевую функцию ограничений, как штрафа. Это позволяет существенно снизить вычислительные затраты по сравнению с PG, при этом ослабив накладываемые ограничения.

Не менее известный подход в RL демонстрируют методы Actor-Critic, особенностью которых является использование Actor'а для выбора действий агентом при взаимодействии со средой с последующим оцениванием их полезности через функцию полезности состояния через Critic'а. Алгоритм Asynchronous Advantage Actor-Critic (A3C) [25] является модификацией метода Advantage Actor-Critic (A2C) и позволяет использовать параллелизм агентов и сред для этапа обучения, в результате получая усредненный результат обучения агентов.

Deep Deterministic Policy Gradients (DDPG) [26] – совмещает идеи методов Actor-Critic и подходов Deep Learning, путем формирования нейронных сетей Actor и Critic. Для повышения стабильности обучения модели и

процесса исследования агентом среды задачи, авторами статьи предлагается использование метода Replay Memory (RM) с целевыми сетями Actor-Critic и ϵ-жадной стратегии.

Глава 1. Математическая постановка и данные

1.1 Модель энергосистемы

В рамках данной работы рассматривается энергосистема, состоящая из фотоэлектрической (солнечной) электростанции, аккумуляторной батареи (накопитель энергии), нагрузки системы (потребители энергии). Солнечная электростанция вырабатывает энергию, которая в дальнейшем используется для удовлетворения спроса на энергию в системе. В случае невозможности удовлетворения нагрузки, спрос может быть удовлетворен путем покупки энергии у местной коммунальной сети в неограниченном объеме. Избыточная энергия в системе, либо продается обратно той же коммунальной сети, либо повторно используется путем ее хранения в имеющейся аккумуляторной батарее. В качестве накопителя выступают литий-ионные батареи в силу их популярности и частоты использования в энергосистемах, благодаря высокому соотношению вместимости энергии и медленной потере заряда в режиме холостого хода.
В момент времени t система управления обладает информацией на:

- тарифы цен на куплю/продажу энергии у местной коммунальной сети (на следующие 24 часа)
- прогнозные значения спроса на энергию и ее выработки (на следующие 24 часа)
- характеристики аккумуляторной батареи

В данной работе имеют место следующие допущения:

- отсутствие эффекта деградации (старения) батареи
- отсутствие влияния окружающей среды на процесс работы батареи
- неизменчивость характеристик батареи с течением времени (емкости, мощности, эффективности)
- на каждом временном шаге над батареей может быть выполнено лишь 1 действие (зарядка/разрядка/холостой ход)

- наличие погрешностей в прогнозных значениях нагрузки и выработки энергии на следующие 24 часа
- горизонт планирования 24 часа, временной интервал 15 минут

Обозначение	Описание
	Параметры
η_c, η_d	эффективность зарядки/разрядки батареи
P_c^{max}, P_d^{max}	максимальная мощность зарядки/разрядки батареи
B^{min}, B^{max}	минимальное/максимальное состояние заряда батареии
$C_b(t), C_s(t)$	тариф на покупку/продажу энергии
$E_l(t)$	прогнозируемая жилая нагрузка
$E_{pv}(t)$	прогнозируемая мощность фотоэлектрической станции
B_0	начальное состояние заряда батареи
	Переменные
$E_b(t), E_s(t)$	приобретенная/проданная энергия
$B(t)$	остаточная мощность в батарее
$P_c(t), P_d(t)$	мощность зарядки/разрядки батареи
$u(t), \nu(t)$	1, если батарея разряжается, 0 иначе

Таблица 1: Параметры и переменные.

Учитывая обозначения, указанные в таблице 1, можно определить целевую функцию задачи следующим образом:

$$F(t) = \sum_{t=0}^{95} E_b(t) \cdot C_b(t) + \sum_{t=0}^{95} E_s(t) \cdot C_s(t) \to min$$

Первая сумма – это стоимость импортируемой энергии из местной коммунальной сети за 24 часа. Второе слагаемое отвечает за прибыль экспортируемой энергии. Теперь необходимо определить ограничения рассматриваемой задачи. Во-первых, ограничение энергитического баланса, смысл которого в необходимости удовлетворения потребности сети в электроэнергии за счет возобновляемого источника энергии, аккумуляторной батареи и обмена энергией с местной коммунальной сетью:

$$E_b(t) + E_{pv}(t) - E_d(t) = E_l(t) + E_c(t) - E_s(t), \quad \forall t.$$
$$E_d(t) \leq 0, E_s(t) \leq 0, \quad \forall t.$$

Связь уровня энергии в аккумуляторной батарее на текущем и предыдущем шаге определяется следующим равенством:

$$B(t) = B(t-1) + P_c(t) \cdot \eta_c + P_d(t)/\eta_d \quad \forall t, t \neq 1$$
$$B(0) = B_0$$

Также необходимо, чтобы удовлетворялись ограничения накладываемые на скорость зарядки/разрядки аккумуляторной батареи, ее вместимость и возможность работы только лишь в одном из 3х режимов:

$$0 \leqslant P_c(t) \leqslant u(t) \cdot P_c^{max} \qquad \nu(t) \cdot P_d^{max} \leqslant P_d(t) \leqslant 0 \quad \forall t,$$
$$B^{min} \leqslant B(t) \leqslant B^{max} \qquad u(t) + \nu(t) \leqslant 1 \qquad \forall t.$$

При решении данной задачи используются прогнозные значения вырабатываемой солнечной электростаницей энергии $E_{pv}(t)$ и жилой нагрузки системы $E_l(t)$, в которых согласно ранее упомянутому допущению, присутствуют ошибки при прогнозировании, используемые при построении ограничения энергитического баланса системы. Ввиду чего, решая задачу смешанным целочисленным линейным программированием, пользователь сталкивается с накапливающимися ошибками в итоговых результатах.

1.2 Обзор данных

Используемые в рамках задачи исторические данные можно разбить на 3 основных категории:

- характеристики накопителя (аккумуляторной батареи)
- данные для обучения модели (460800 наблюдений)
- данные для тестирования модели (115200 наблюдений)

Данные обучения и тестирования поделены на 11 случаев. Каждому случаю соответствует файл с наблюдениями случайного числа периодов (временных отрезков) разной продолжительности, измеряемой в днях и минимальная длина которых составляет 10 дней. Временные отрезки (периоды) никак не связаны между собой, не пересекаются и разбросаны случайным

образом в период с 2014 по 2018 год. Каждое наблюдение несет в себе следующую информацию:

- номер случая
- номер периода (временного отрезка)
- временной шаг ‘год-месяц-день-часы-минуты’
- потребление и выработка энергии на предыдущем временном шаге (на предыдущих 15 минутах)
- прогнозные значения потребления и выработки энергии (на следующие 24 часа)
- точные значения тарифов на куплю/продажу энергии у местной коммунальной сети (на следующие 24 часа)

Test case	B^{max}	B^{min}	P_c^{max}	P_d^{max}	η_d	η_c
1	300	0	75	-75	0.950	0.950
2	600	0	150	-150	0.950	0.950
3	100	0	25	-25	0.950	0.950
4	100	0	25	-25	0.950	0.950
5	10	0	2.5	-2.5	0.950	0.950
6	1500	0	375	-375	0.950	0.950
7	400	0	100	-100	0.950	0.950
8	10	0	2.5	-2.5	0.950	0.950
9	100	0	25	-25	0.950	0.950
10	300	0	75	-75	0.950	0.950
11	600	0	150	-150	0.950	0.950

Таблица 2: Характеристики аккумуляторной батареи

В таблице 2 отображены характеристики аккумуляторной батареи, а конкретно информация о емкости, мощности, эффективности зарядки и разрядки батареи.

На рисунке 1 визуализировано выполнение условия энергетического баланса системы и количество необходимой для этого энергии. Это в свою

очередь толкает на необходимость покупки энергии у местной коммунальной сети, поскольку для удовлетворения спроса на энергию недостаточно лишь ее выработки за счет солнечной электростанции.

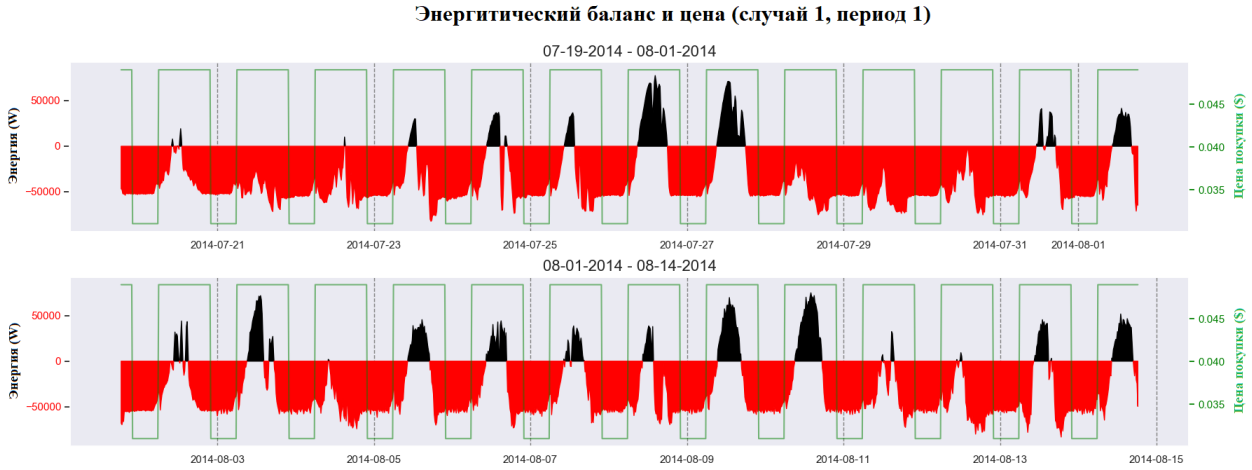


Рис. 1: Примеры энергитического баланса

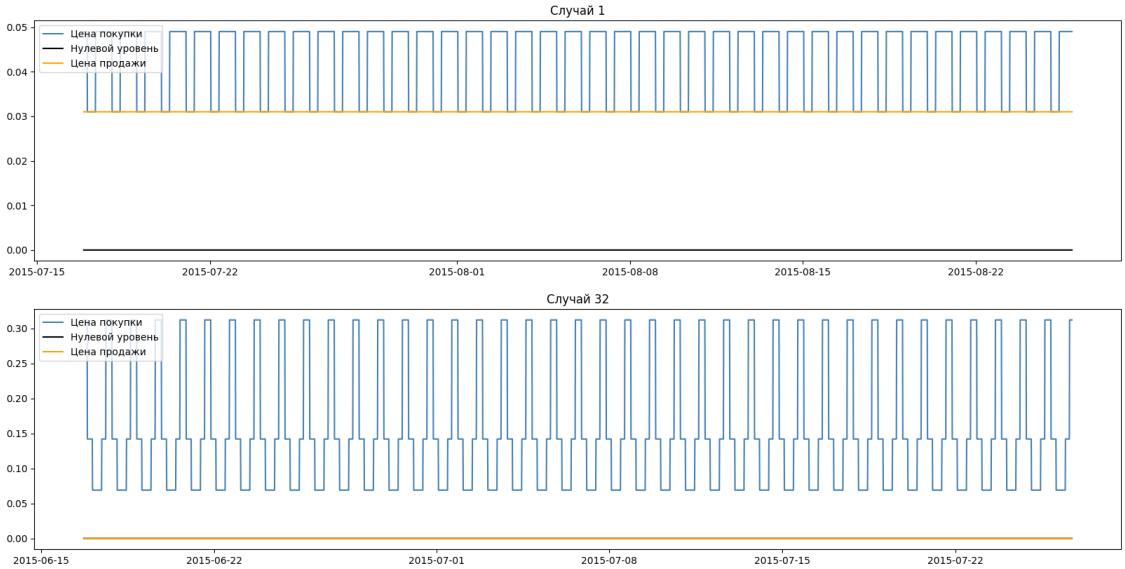


Рис. 2: Пример тарифов на продажу/покупку энергии у местной коммунальной сети

Из графиков на тарифы местной коммунальной сети (рис. 2) можно сделать вывод о небходимости наличия достаточно гибкой модели или же формирования ряда моделей для каждого случая отдельно с целью получения более удовлетворительных результатов, поскольку продажа лишней энергии по нулевой цене никак не позволит минимизировать затраты.

Далее представим графики сравнения актуальных и прогнозных значений нагрузки и выработки PV (рис. 3-4).

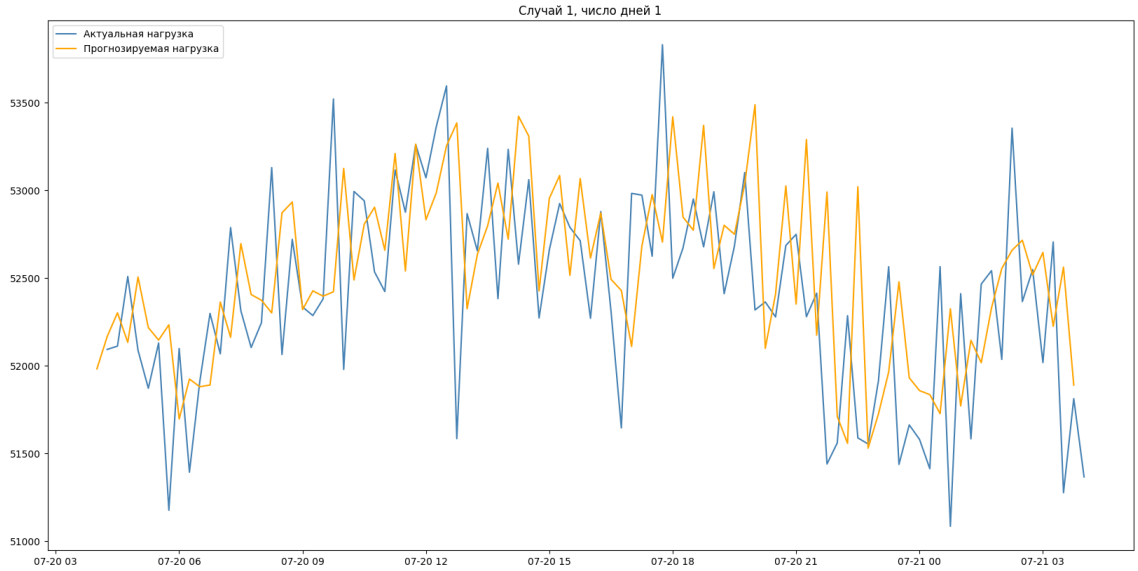


Рис. 3: Сравнение актуальных и прогнозных значений нагрузки

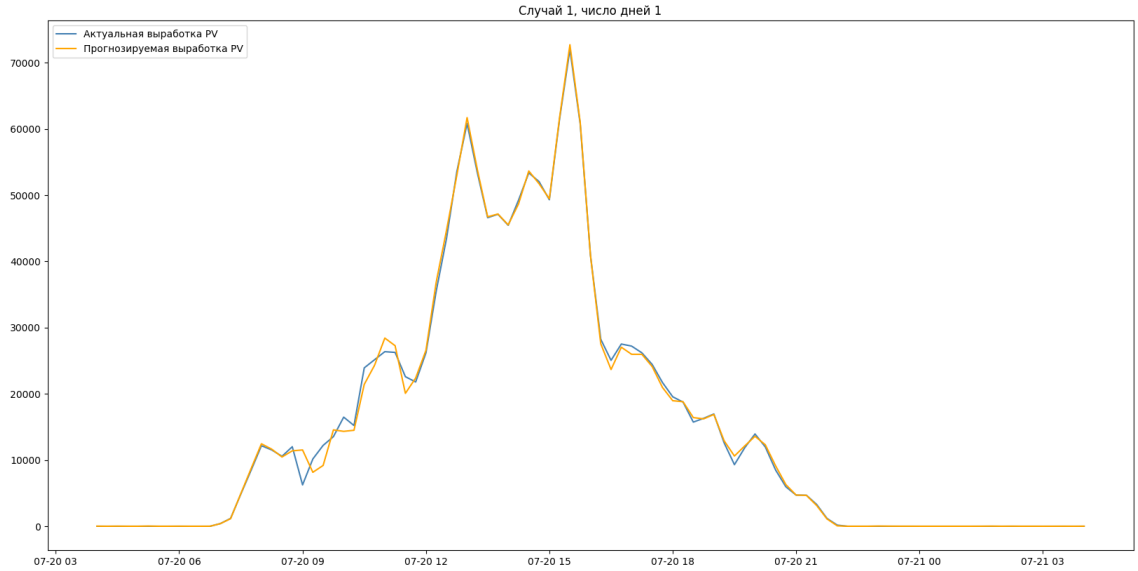


Рис. 4: Сравнение актуальных и прогнозных значений выработки PV

Для оценки точности прогнозирования предлагается в качестве метрики использовать взвешенную абсолютную процентную ошибку, являю-

щейся малочувствительной к наличию выбросов:

$$WAPE(y, \widehat{y}) = \frac{\sum_{i=1}^{T} |y_i - \widehat{y}_i|}{\sum_{i=1}^{T} y_i},$$

где $y, \widehat{y}$ — фактическое и предсказанное значение энергопотребления или выработки солнечной электростанции соответственно, T – количество шагов для управления в тестовом случае.

Средний процент ошибки WAPE составил 8.86% для нагрузки и 4.45% для выработки PV. При решении поставленной задачи методами стандартными методами линейного программирования приводит к накоплению ошибок и значительному отклонению результатов от оптимального решения, поскольку прогнозируемых параметров 192 из суммарных 391.

Ниже представлена таблица результатов численного эксперимента (см. таб. 3), где *score* имеет смысл среднего значения относительных финансовых затрат и вводится в главе 3. Данная метрика была рассчитана с использованием имеющихся в исторических данных прогнозных значений, в то время как $score_{adj}$ несет тот же смысл, но прогнозы заменены их актуальными значениями только для текущего временного шага.

Случай	*score*	$score_{adj}$	1 % score = x % ошибки прогноза
1	0.048	0.049	1.16
2	0.084	0.086	0.43
3	0.164	0.171	0.14
4	0.185	0.204	0.06
5	0.109	0.123	0.07
6	0.185	0.192	0.15
7	0.191	0.215	0.04
8	0.205	0.240	0.03
9	0.113	0.131	0.06
10	0.112	0.121	0.12
11	0.165	0.180	0.1
AVG	0.142	0.156	0.22

Таблица 3: Численные эксперименты с использованием метода GLOP

В следствии чего можно определить насколько сильно наличие ошибки

прогнозной модели влияет итоговый на результат при решении задачи смешанным целочисленным линейным программированием. На основе предоставленных результатов видно, что для некоторых случаях отслеживаются существенные отклоенения показателя $score$ от $score_{adj}$ при наличии ошибок в прогнозе, что говорит о необходимости использования более точного прогнозного модуля при использовании MILP. При необходимости планирования на час вперед (вместо 15 минут), представленные выше показатели начнут существенно возрастать, что затрудняет использование данного подхода в целях получения оптимальных результатов в задачах с долгосрочным планированием. Другим не менее важным недостатком является необходимость в повторном решении задачи в случаях изменения входных значений, например, процесс деградации аккумуляторной батареи.

В следующей главе предлагается рассмотреть подход и методы обучения с подкреплением, которые позволяют устранить имеющиеся при использовании MILP недостатки при решении исследуемой задачи.

Глава 2. Обучение с подкреплением

2.1 Мотивация

Основными преимуществами в использовании подхода обучения с подкреплением в данной задаче являются: возможность обученного агента быстро принимать необходимо решение касательно управления, умение подстраиваться под задачу в случае изменения входных значений путем обучения на новых данных, возможность работы с неопределенностями в исторических данных. Алгоритмы RL дают возможность обучить агента решать конкретную задачу без надобности в формировании и использовании модели рассматриваемой проблемы, работая лишь с историческими данными. При возникновении непредвиденных случаев агент за счет возможности к адаптации с течением времени сможет выполнять действия, наиболее ценные с точки зрения получаемой им суммарной награды.

2.2 Общие понятия обучения с подкреплением

Марковский процесс принятия решений (МППР) предназначен для прямой постановки проблемы обучения, базирующегося на взаимодействии для достижения цели. Обучающийся и принимающий решения называется агентом, а то, с чем он взаимодействует, включая все что находится за его пределами, называется средой.

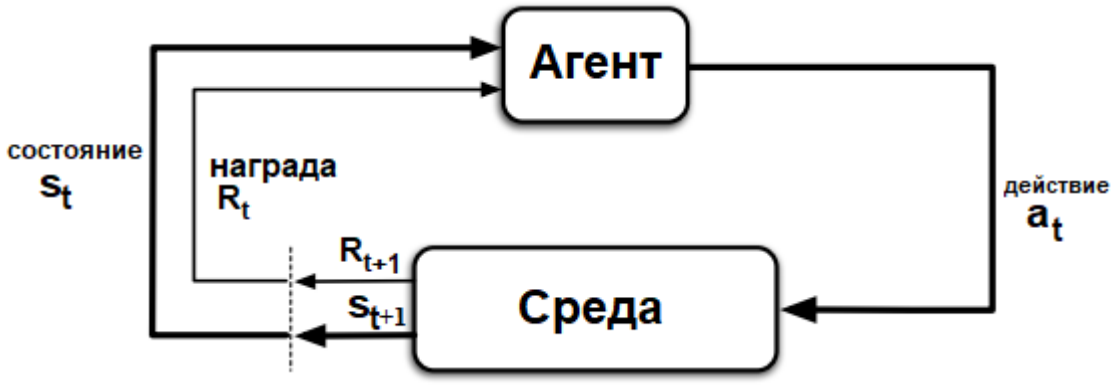


Рис. 5: Взаимодействие агент-среда в марковском процессе принятия решений

Агент, постоянно взаимодействуя со средой, выбирает действия, а

среда реагирует на них и преставляет обучающемуся новые ситуации. Также среда формирует награду, числовое значение, которое агент стремится максимизировать с течением времени путем выбора действий.

Взаимодействие агента и среды происходит на каждом из последовательных дискретнных временных шагов $t = 0, 1, 2, 3, ...$. В любой момент времени t агент получает информацию о состоянии среды s_t ($s_t \in \mathcal{S}$), и основываясь на этом выбирает действие a_t ($a_t \in \mathcal{A}(s)$). Как последствие своего действия a_t агент получает численную награду R_{t+1} ($R_{t+1} \in \mathcal{R} \subset \mathbb{R}$) – реализацию случайной величины, как правило зависящую от s_{t-1} и a_{t-1}, и информацию о новом состоянии s_{t+1}.

В данной работе будет рассматриваться конченый марковский процесс принятия решений – когда множества состояний, действий и наград $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ являются конечными множествами.

2.2.1 Марковский процесс принятия решений

Перед тем, как ввести понятие марковского процесса принятия решений, необходимы несколько вспомогательных определений и свойств.

Известно из [18], что некоторое состояние s_t обладает марковским свойством $\Longleftrightarrow$ когда выполняется:

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, ..., s_t],$$

Тогда вероятность перехода между состояниями s и s' определяется:

$$\mathcal{P}_{ss'} = \mathbb{P}[s_{t+1} = s'|s_t = s],$$

Матрица вероятностей переходов $\mathcal{P}$ (в случае если $\mathcal{S} = \{1, 2, ..., n\}$) определяет вероятности перехода из всех состояний s во все состояния s':

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & ... & \mathcal{P}_{1n} \\ ... & ... & ... \\ \mathcal{P}_{n1} & ... & \mathcal{P}_{nn} \end{bmatrix},$$

где $\sum_{j=1}^{n} \mathcal{P}_{ij} = 1$ $(\forall j \in \overline{1, n})$.

Марковский процесс с вознаграждением – это кортеж $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, для которого случайные состояния из последовательности $\{s_1, s_2, ...\}$ обладают марковским свойством, где:

- $\mathcal{S}$ – пространство состояний ($\forall$s $\in \mathcal{S}$)
- $\mathcal{P}$ – матрица переходных вероятностей, для которой выполняется $\mathcal{P}_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s]$ ($\forall s, s' \in \mathcal{S}$)
- $\mathcal{R} \subset \mathbb{R}$ – множество всех вознаграждений, а функция вознаграждения $\mathcal{R}_s = \mathbb{E}[R_{t+1} | s_t = s]$ ($\forall s \in \mathcal{S}$)
- γ – коэффициент дисконтирования (оценивает ценность будущих наград), $\gamma \in [0, 1]$

В случае конечного марковского процесса, случайные величины R_t и s_t имеют дискретное вероятностное распределение, зависящее лишь от предыдущего состояния и действия. В таком случае, для конкретных случайных величин, $s' \in \mathcal{S}$ и $r \in \mathcal{R}$, можно определить вероятность p ($p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1]$) их появления в момент времени t, имея предыдущие состояние s и действие a:

$$p(s', r | s, a) = \mathbb{P}[s_t = s', R_t = r | s_{t-1} = s, a_{t-1} = a],$$

что верно для $\forall s, s' \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$.
При этом место следующее равенство для $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1.$$

В таком случае можно посчитать ожидаемую награду $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$:

$$r(s, a) = \mathbb{E}[R_t | s_{t-1} = s, a_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

И ожидаемую награду для случая $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$:

$$r(s, a, s') = \mathbb{E}[R_t | s_{t-1} = s, a_{t-1} = a, s_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}$$

Под совокупной наградой G_t будем понимать следующую сумму:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{i=0}^{T-t} \gamma^i R_{t+i+1}$$

где T – это последний шаг (горизонт). Необходимость в нем появляется, когда в задаче имеется естественное понятие конечного временного шага, а именно когда взаимодействие агента с окружающей средой можно разбить на подпоследовательности, называемые эпизодами, например, путешествие по лабиринту.
В случае отсутствия у задачи горизонта совокупная награда имеет вид:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}$$

G_t является конечной суммой [18] в том случае, если награда не равна нулю, есть величина постоянная и $\gamma < 1$.
Под функцией полезности состояния $v(s)$ будем понимать:

$$v(s) = \mathbb{E}[G_t | s_t = s]$$

В [18] было показано, что используя понятие функции полезности можно в результате некоторых преобразований прийти к Bellman Expectation Equation:

$$v(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | s_t = s] = \sum_{s', r} p(s', r|s, a)[r + \gamma v(s') | s_t = s],$$

где r это непосредственная награда ($r \in \mathcal{R}$).
С учетом ранних обозначений $\mathcal{R}_s$ и $\mathcal{P}_{ss'}$ можно придти к следующему виду:

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Тогда в матричном виде оно имеет вид $v = \mathcal{R} + \gamma \mathcal{P} v$:

$$\begin{bmatrix} v(1) \\ ... \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ ... \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & ... & \mathcal{P}_{1n} \\ ... & ... & ... \\ \mathcal{P}_{n1} & ... & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ ... \\ v(n) \end{bmatrix}$$

Данное уравнение имеет прямое решение $v = (E - \gamma \mathcal{P})^{-1} \mathcal{R}$, но лишь в случаях небольших марковских процессов с наградой. При работе с большими марковскими процессами применяется либо динамическое программирование, либо методы Монте-Карло [18].

Конечный марковский процесс принятия решений – это кортеж вида $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ – конечное множество состояний, $\forall$s $\in \mathcal{S}$
- $\mathcal{A}$ – конечное множество действий, $\forall$a $\in \mathcal{A}(s)$
- $\mathcal{P}$ – матрица переходных вероятностей, для которой выполняется $\mathcal{P}_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a]$, $\forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}(s)$
- $\mathcal{R} \subset \mathbb{R}$ – множество всех вознаграждений, а функция вознаграждения $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | s_t = s, a_t = a]$ для $\forall s \in \mathcal{S}$ и $\forall a \in \mathcal{A}(s)$.
- γ – коэффициент дисконтирования, $\gamma \in [0, 1]$

Под статегией (политикой) π будем понимать вероятность действия при заданном состоянии:

$$\pi(a|s) = \mathbb{P}[a_t = a | s_t = s]$$

Она определяет процессом принятия решений агентом (его поведение), зависит только от текущего состояния и стационарна.

При заданном МППР $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ и, используя понятие стратегии π, можно определить вероятность перехода из состояния s в состояние s'

согласно политике π, а также функцию награды политки:

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^{a}_{s},$$

где $\pi(a|s)$ – это вероятность выбора действия a при условии пребывания в состоянии s в соответствии со стратегией π.
В таком случае Bellman Expectation Equation, используя понятие политики π, для функции полезности состояния имеет вид [18]:

$$v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | s_t = s] = \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\pi}(s')|s_t = s]$$

Под функцией полезности действия $q_{\pi}(s, a)$ марковского процесса принятия решений будем понимать:

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

По аналогии с функцией полезности состояния можно получить Bellman Expectation Equation для функции полезности действия $q_{\pi}(s, a)$:

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | s_t = s, a_t = a]$$

$$q_{\pi}(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma q_{\pi}(s', a')|s_t = s]$$

$$q_{\pi}(s, a) = \mathcal{R}^{\pi}_{s} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\pi}_{ss'} q_{\pi}(s', a')$$

В [18] было показано, что как и в случае марковского процесса принятия решения без стратегии π, можно получить Bellman Expectation Equation в матричном виде для $v_{\pi}(s)$ и $q_{\pi}(s, a)$ и найти их решение:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$
$$q_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi q_\pi$$

Соответствующие им прямые решения имеют вид $v_\pi = (E - \gamma\mathcal{P}^\pi)^{-1}\mathcal{R}^\pi$ и $q_\pi = (E - \gamma\mathcal{P}^\pi)^{-1}\mathcal{R}^\pi$. Здесь E это единичная матрица размерности $|\mathcal{S}| \times |\mathcal{S}|$.

Под оптимальной функцией полезности состояния относительно всех стратегий будем понимать следующее:

$$v^*(s) = \max_\pi v_\pi(s)$$

Аналогично можно ввести понятие оптимальной функции полезности действия относительно всех стратегий:

$$q^*(s, a) = \max_\pi q_\pi(s, a)$$

Введем введем отношение частичного порядка для политик:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s), \forall s$$

В [18] была доказана теорема, которая утверждает:

- существует оптимальное π^*, для которого выполняется $\pi^* \geq \pi$, $\forall \pi$
- при всех оптимальных стратегиях π^* достигается оптимальная функция полезности состояния $v_{\pi^*}(s) = v^*(s)$
- при всех оптимальных стратегиях π^* достигается оптимальная функция полезности действия $q_{\pi^*}(s, a) = q^*(s, a)$

Решение Bellman Optimization Equations для $v_\pi(\cdot), q_\pi(\cdot, \cdot)$ позволило бы получить $v^*(\cdot), q^*(\cdot, \cdot)$, однако, в силу их нелинейности и отсутствия аналитического решения в общем случае, предлагается использовать следующие итеративные подходы:

- динамическое программирование (Value Iteration, Policy Iteration)
- Temporal Difference Learning (Q-learning)

Основным недостатком простого динамического программирование является необходимость в полном знании марковского процесса принятия решений (вероятностей переходов), а также представления функций полезности состояния $v(\cdot)$ и действия $q(\cdot, \cdot)$. Поскольку зачастую при решении реальных индустриальных задач мы не располагаем такой информацией, возникает идея аппроксимировать данные функции. Задачей таких методов является получение политики π близкой к оптимальной. Это в свою очередь обязывает следить за величиной следующих ошибок:

- sampling/estimation error
- approximation error

2.2.2 Policy Iteration и Value Iteration

Дальнейшие суждения верны только для среды, являющейся конечным марковским процессом принятия решений.

Основной принцип работы Policy Iteration заключается в оценка стратегии и ее улучшении. В самом начале случайным образом происходит инициализация детерминированной стратегии $\pi(s) \in \mathcal{A}(s)$ и оценок $V(s) \in \mathbb{R}$ $(\forall s \in \mathcal{S})$. Поскольку множество состояний $\mathcal{S}$ конечно, небходимо определить терминальное состояние и/или длину горизонта $(V(terminal) = 0)$. На рисунке 6 продемонстрирован принцип работы данного алгоритма, согласно которому после получения оценочных значений для всех состояний системы происходит улучшение политики за счет какого-либо подхода (например жадного).

В [18] описан принцип оптимальности, согласно которому стратегия $\pi(a|s)$ достигает оптимальной функции полезности из s $(v_\pi(s) = v^*(s)) \Longleftrightarrow$ когда:

- любое состояние s' достижимо из s
- π достигает оптимального значения из состояния s' $(v_\pi(s') = v^*(s'))$

Сходимость данного итерационного процуесса к оптимальной стратегии (оптимальной функции полезности) за конечное число итераций обеспечивается за счет конечности МППР и свойств итерационного процесса

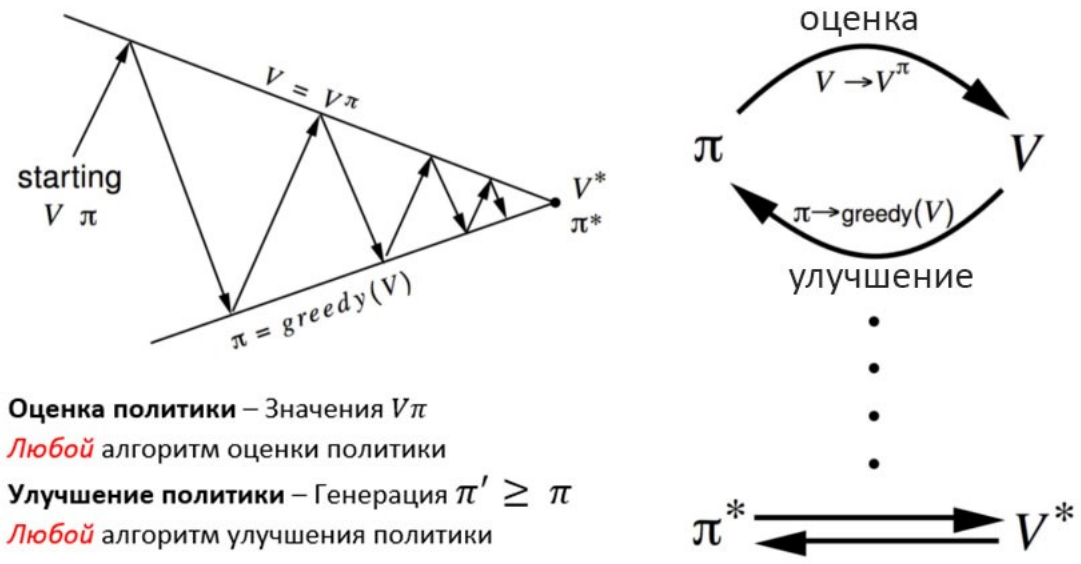


Рис. 6: Принцип работы policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\mathit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Рис. 7: Алгоритм метода Policy Iteration

Беллмана. В случае прекращения улучшения стратегии полученная функ-

ция полезности будет удовлетворять уравнению Беллмана [18]:

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a),$$

а значит $v_\pi(s) = V^*(s)(\forall s \in \mathcal{S})$, и π это оптимальная стратегия.

Метод Value Iteration, в отличие от Policy Iteration, использует для обновления значений оценки не ожидаемое значение, а наибольшее, снижая при этом вычислительные затраты.

$$V(s) \leftarrow \max_{a \in \mathcal{A}} (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s'))$$

2.2.3 Temporal Difference Learning

TD Learning – это комбинация идей методов Монте-Карло и динамического программирования. Подобно методам Монте-Карло данных подход позволяет проводить обучение непосредственно на исходных данных, без прибегания к вероятностям перехода марковского процесса принятия решений. Аналогично динамическому программированию происходит обновление прогнозируемых (ожидаемых) значений без ожидания завершения эпизода за счет использования bootstrap (т.е. основываясь на оценочных значениях, а не точных). Главной его особенностью является использование оценки функции полезности $v_\pi(s)$ при заданной стратегии π для поиска оптимальной стратегии.

Обновление функции полезности в данном случае имеет вид:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

Оно происходит на следующем шаге за счет использования награды R_{t+1} и оценки $V(s_{t+1})$. Для этого используется так называемая TD-Error (ошибка временной разницы), которую принято обозначать [18], как δ_t. Ее смысл в разнице между оцененным значением $V_(s_t)$ и улучшенной оценкой $[R_{t+1} + \gamma V(s_{t+1})]$:

$$\delta_t = [R_{t+1} + \gamma V(s_{t+1})] - V(s_t)$$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Рис. 8: Алгоритм метода TD(0) для оценивания v_π

Q-learning

Данный алгоритм относится к классу off-policy learning, что значит для поиска оптимальной стратегии агент не использует политику, исследуя среду (например, случайными действыиями). Подход on-policy learning заключается в исследовании среды посредству текущей политики, которую мы стараемся улучшить. Распространенным подходом для методов off-policy является формирование двух стратегий: для генерации поведения (behavior policy) и целевая (target policy), которая будет улучшаться и исследоваться в процессе обучения.

Метод Q-learning является одним из самых ранних открытий в области обучения с подкреплением и основан на следующем итерационном процессе:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Все значения Q для всех пар (s, a) сохраняются в специальную таблицу. Как и в случае с Value Iteration, данный алгоритм позволяет аппроксимировать значение $q^*(\cdot, \cdot)$. Минимальным требованием сходимости метода в общем является обновление значений для всех пар (s, a) [18].

Однако, у данного метода имеется недостаток: в случаях большой размерности пространств $\mathcal{S}, \mathcal{A}$ использование таблиц для хранения значе-

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Рис. 9: Алгоритм метода Q-learning

ний Q будет очень затратным, ввиду чего используются алгоритмы Policy Gradient и методы из области глубокого обучения с подкреплением (Deep Reinforcement Learning).

2.2.4 Policy Gradient

Рассмотрим класс методов, который позволит отойти от построения распределения состояний-действий Q и выбирать действия не основавыясь на построенной оценке. Введем понятия параметрической стратегии, которая позволит выбирать действия, используя значения Q только при оценке параметров стратегии.

Пусть $\theta \in \mathbb{R}^n$ – вектор параметров стратегии, тогда вероятность выбора действия a в момент времени t в случае, если среда находится в состоянии s с набором параметров θ можно определить следующим образом:

$$\pi(a|s, \theta) = \mathbb{P}\{a_t = a|s_t = s, \theta_t = \theta\}$$

По аналогии с функцией полезности состояния для политики получаем $v(s, w)$, где w вектор весов состояний ($w \in \mathbb{R}^{\tilde{n}}$).

Поиск стратегии у алгоритмов Policy Gradient выполняется использованием градиента скалярной меры эффективности $J(\theta)$ относительно параметров стратегии. Целью является максимизация эффективности, поэтому

используется градиентный подъем по $J(\theta)$:

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t)$$

где $\hat{J}(\theta_t) \in \mathbb{R}^n$ – стохастическая оценка, математическое ожидание которой аппроксимирует градиент меры эффективности относительно θ.

Параметризация стратегии при решении задач методом Policy Gradient может быть проведена любым образом при условии, что полученная $\pi(a|s,\theta)$ является дифференциируемой относительно своих параметров, существует и конечно $\nabla\pi(a|s,\theta)$ $(\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \forall \theta \in \mathbb{R}^{\tilde{n}})$. В работе [18] было показано, что для исследуемости пространства действий необходимо ограничение $\pi(a|s,\theta) \in (0,1)$ $(\forall s, a, \theta)$.

Существуют алгоритмы, которые аппроксимируют не только меры эффективности, но и функцию полезности $v(\cdot)$ – методы Actor-Critic. Задачей Actor является выбор конкретных действий (стратегия), в то время как Critic оценивает полезность выбранных действий через функцию полезности состояния $v(\cdot)$.

2.2.5 Actor-Critic

В основе алгоритма легло использование $\hat{v}(s,w)$ для обновление параметров стратегии $\pi(a|s,\theta)$:

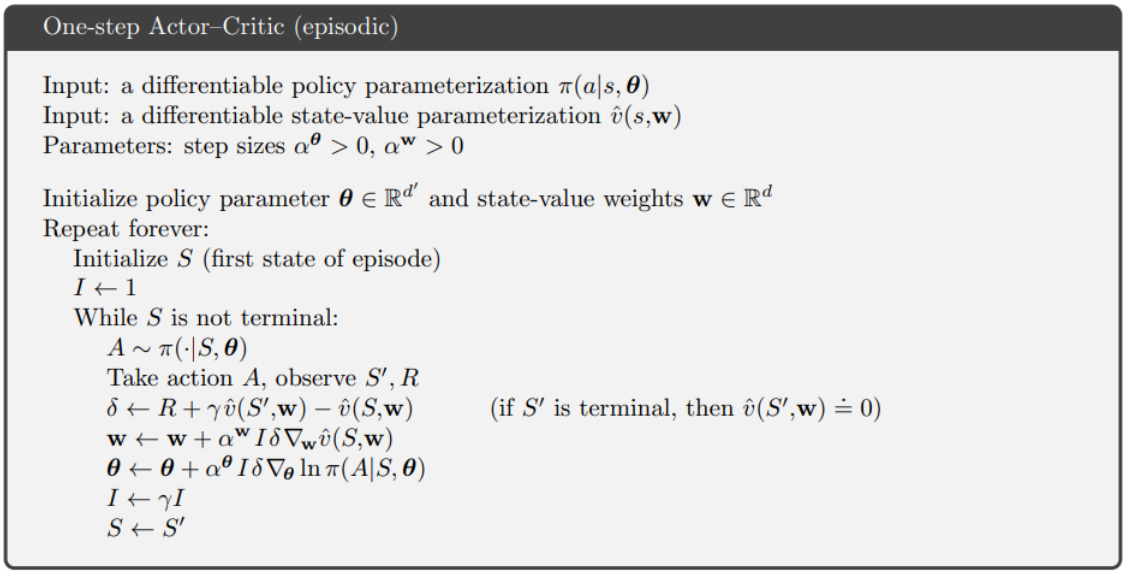
$$\theta_{t+1} = \theta_t + \alpha((G_{t+1} - G_t) - \hat{v}(s_t,w))\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$$

$$= \theta_t + \alpha((r_{t+1} - \gamma\hat{v}(s_{t+1},w)) - \hat{v}(s_t,w))\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$$

$$= \theta_t + \alpha\delta_t\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$$

Здесь δ_t – TD-Error из метода Temporal Difference Learning, которую можно также интерпретировать, как значение ценности действия выбранного из состояния s_t. Градиент от политики позволяет выбрать направление при котором вероятность выбрать действие a_t в состоянии s_t будет наибольшей,

позволив таким образом максимизировать награду [27]. Ниже представлен псевдокод алгоритма Actor-Critic (рис. 10), в котором отношение $\frac{\nabla\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$ записано как $\nabla ln\pi(a_t|s_t,\theta_t)$.



One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I\delta\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla_{\boldsymbol{\theta}} \ln\pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Рис. 10: Эпизодический одношаговый алгоритм Actor-Critic

2.3 Алгоритмы обучения с подкреплением

2.3.1 Proximal Policy Optimization (PPO)

Основное отличие данного метода от Policy Gradient заключается в подходе к обновлению параметров политики: вместо строгого ограничения в целевую функцию вводится штраф с весовым коэффициентом. Это позволяет ослабить условия задачи при этом снизив вычислительные затраты. В качестве оптимизируемой целевой функции рассматривается ожидаемая награда – обозначим ее как $\eta(\theta)$.

Алгоритм использует идею итеративного метода оптимизации MM algorithm (Majorize-Minimization, Minorize-Maximization) [28], суть которого заключается в построении некоторой функции $g(\theta|\theta_m)$, которая бы удовлетворяла следующим условиям:

$$g(\theta|\theta_m) \leq \eta(\theta), \quad \forall\theta,$$
$$g(\theta_m|\theta_m) = \eta(\theta_m),$$

где $\eta(\cdot)$ должна быть вогнутой функцей, если решается задача максимизации. В работе [29] была доказана сходимость данного итерационного процесса к локальному оптимуму при стремлении m к бесконечности.

В качестве метода оптимизации PPO используется trust region [30], поскольку градиентный спуск может испортить процесс обучения модели RL неудачным выбором длинны шага. В качестве максимальной длины шага trust region предлагается брать следующую величину:

$$D_{KL}(\pi_{new}||\pi_{old}) = \mathbb{E}_\theta[log\frac{\pi_{new}(\theta)}{\pi_{old}(\theta)}],$$

здесь D_{KL} – дивергенция Кульбака–Лейблера, измеряющая меру различия старой и новой политик.

В работе [30] была получена нижняя граница для функции ожидаемой награды $\eta(\cdot)$:

$$\begin{aligned}
&\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - CD_{KL}^{max}(\pi, \tilde{\pi}),\\
&D_{KL}^{max}(\pi, \tilde{\pi}) = \max_s[D_{KL}(\pi(\cdot|s)||\tilde{\pi}(\cdot|s))],\\
&C = \tfrac{4\epsilon\gamma}{(1-\gamma)^2}, \qquad \epsilon = \max_{s,a}|A_\pi(s,a)|, \qquad L_\pi = \hat{\mathbb{E}}_t\left[\tfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right],\\
&A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s).
\end{aligned}$$

L_π функция потерь, A_π функция преимущества, $\hat{A}_t$ оценка функции преимущества. Главным недостатком является строгость, накладываемая оператором максимума от дивергенции. Перейдем от максимума дивергенции к оценке ее ожидаемого значения:

$$\begin{aligned}
&\max_\theta \hat{\mathbb{E}}_t\left[\tfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right],\\
&s.t. \quad \hat{\mathbb{E}}_t\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))\right] \leq \delta,
\end{aligned}$$

где δ некоторое значение, которое в дальнейшем будем считать гиперпараметром. Перепишем задачу, записав ограничение в виде штрафа целевой функции:

$$\max_\theta\left(\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t))\right]\right),$$

где β это весовой коэффициент штрафа. В [30] было показано, что решением такой задачи будет:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T F^{-1} g}} F^{-1} g,$$
$$g \doteq \nabla_\theta L_{\theta_k}(\theta)|_{\theta_k}, \qquad F \doteq \nabla^2_\theta \hat{D}_{KL}(\theta||\theta_k)|_{\theta_k}.$$

Возникает проблема в необходимости подсчета обратной матрицы вторых производных, что является очень затратными вычислениями. Метод PPO избегает это за счет формирования решения в первых производных (например градиентный спуск) достаточно близким к решению во вторых производных путем добавления слабых ограничений. В результате чего имеется вероятность рано или поздно получить плохую политику, используя градиентный спуск, но уменьшив при этом вероятность ее появления за счет добавления слабых условий в целевую функцию.

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

Рис. 11: PPO с адаптивным штрафов дивергенции Кульбака–Лейблера

2.3.2 Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Advantage Actor-Critic (A3C) – модификация метода Advantage Actor-Critic (A2C). Главная идея основана на методам Actor-Critic, однако для улучшения сходимости используются параллельно несколько агентов и сред, в следствие чего для получения итогового результата

полученные промежуточные вычисления усредняются в глобальной сети. Поэтому далее будем рассматривать схему и алгоритм работы метода A2C.

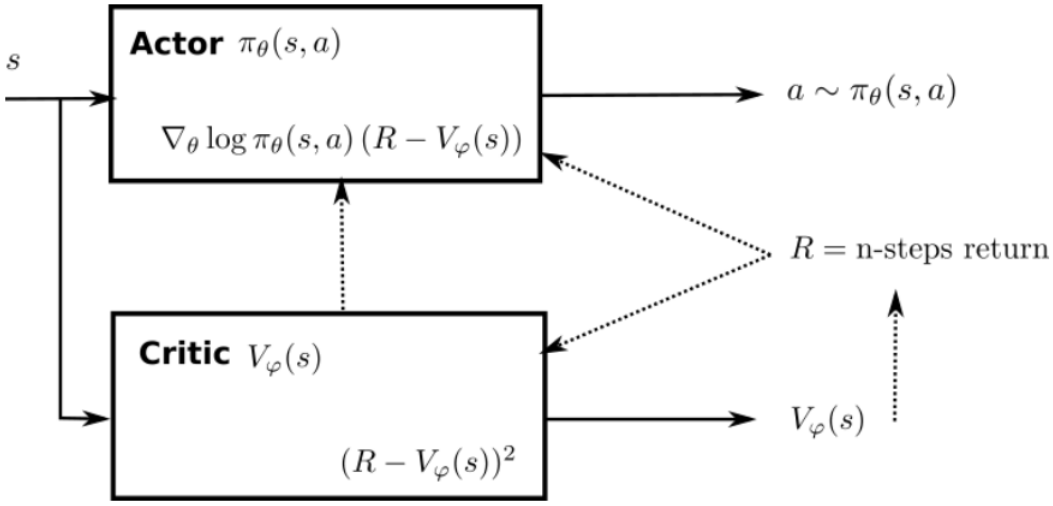


Рис. 12: Принцип работы метода A2C

Главное отличие метода A2C от обычного Actor-Critic в том, что первый оценивает функцию полезности состояния v на несколько шагов вперед. Из схемы видно, что задача Actor'а заключается в вычислении параметризованной политики $\pi_\theta(s,a)$ и выборе действий, в том время как задача Critic'а находить значение $v_\varphi(s)$.

Опишем алгоритм работы метода A2C:

1. получить некоторый набор кортежей (s,a,r,s'), используя исходную стратегию π_0

2. для каждого полученного s Actor подсчитывает сумму следующих n наград, critic вычисляет функцию полезности для n имеющихся состояний:

$$R_t = (\sum_{k=0}^{n-1} \gamma^k R_{t+k+1}) + \gamma^n v_\varphi(s_{t+n+1})$$

3. обновить стратегию Actor согласно методу Actor-Critic

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log \pi_\theta(s_t,a_t)(R_t - v_\varphi(s_t))$$

4. обновить Critic с целью минимизации ошибки отклонения (Temporal Difference ошибка)

$$L(\varphi) = \sum_t (R_t - v_\varphi(s_t))^2$$

При этом градиент Actor обновляется по следующей формуле:

$$\theta \leftarrow \theta + \nabla_\theta \log \pi_\theta(s_t, a_t)(R_t - v_\varphi(s_t))$$

В то время, как градиент Critic обновляется:

$$\varphi \leftarrow \varphi + \nabla_\phi (R_t - v_\varphi(s_t))^2$$

2.3.3 Deep Deterministic Policy Gradients (DDPG)

Глубокий детерминированный градиент политики (DDPG) – это метод оценки политики, в котором параметризованная модель основана на нейронной сети глубокого обучения. Структура данного алгоритма использует идею методов Actor-Critic, а именно нейронные сети для формирование behavior политики Actor $Q(s, a|\theta^Q)$ и Critic $\mu(s|\theta^\mu)$ и для target политики Actor $Q'(s, a|\theta^{Q'})$ и Critic $\mu'(s|\theta^{\mu'})$. Использование такого подхода позволяет добиться некоторой стабильности на этапе обучения агента.

Обновление весов target политики происходит по принципу softmax:

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu'} + (1-\tau)\theta^\mu, \qquad \theta^{Q'} \leftarrow \tau\theta^{Q'} + (1-\tau)\theta^Q,$$

где $\tau \in [0, 1]$, чаще всего принимаемый близким к 1.

Сеть Actor в качестве политики выбирает $\mu(s) = \arg\max_a Q(s, a)$. Функции потерь для μ и Q выглядят следующим образом:

$$J_Q = \frac{1}{N}\sum_{i=1}^{N}(R_i + \gamma(1-d)Q'(s_{i+1}, \mu'(s_{i+1})) - Q(s_i, \mu(s_i)))^2$$

$$J_\mu = \frac{1}{N}\sum_{i=1}^{N} Q(s_i, \mu(s_i))$$

Для сети μ функция потерь представляет сумму значений Q для всех состояний $\{s_1, ..., s_N\}$. Для вычисления значений Q используется сеть Critic, которой подается действие $\mu(s_i)$, вычисленное сетью Actor. Задачей является максимизация данной величины с целью получения наибольших значений Q. Для сети Q функция потерь это Temporal Difference ошибка (отклонение от оптимального значения q^*), ввиду чего мы минимизируем данные потери.

Обновление весовых коэффициентов сети Q происходит путем минимизации функции потерь среднеквадратической ошибки. Для сети μ находится производная ее функции потерь и методом back propagation (обратного распространения ошибки), обновляются веса нейронной сети (по вычисленному градиенту):

$$\nabla_{\theta^\mu} J_\mu = \frac{1}{N} \sum_{i=1}^{N} [\nabla_\mu Q(s, \mu(s)) \nabla_{\theta^\mu} \mu(s)]$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Рис. 13: Алгоритм DDPG

В представленном псевдокоде алгоритма (см. рис. 13) используется метод вопроизведения опыта (replay memory) для повышения стабильности обучения. Заключается он в формировании некоторого массива кортежей (s, a, s', r) называемого replay-buffer, которые используются агентом во время обучения. Размер данного массива задается заранее, обучение начинается только в том случае, когда replay memory будет полностью заполнена и при получении новых кортежей (s, a, s', r), самые старые экземпляры заменяются на новые, позволяя поддерживать баланс между старыми кортежами и новыми, тем самым улучшая обучение агента.

Для exploration (исследования) среды агентом алгоритм использует ϵ-жадную стратегию, но в случае с непрерывным пространством действий исследование обеспечивается путем добавления к действию шума (в данном случае шум формируется по принципу процесса) Орнштейна-Уленбека.

Глава 3. Этапы в решении задачи с помощью RL

Для того чтобы приступить к решению какой-либо задачи с помощью Reinforcement Learning, необходимо ответить на следующие важные вопросы:

- как правильно сформировать среду
- каким образом выбирать алгоритм RL
- возможно ли ускорить процесс обучения модели

3.1 Формирование среды

Чтобы сформировать среду, необходимо определить ее основные компоненты, а именно: пространство состояний $\mathcal{S}$, пространство действий $\mathcal{A}$ и множество наград $\mathcal{R}$.

Пространство состояний ($\mathcal{S}$). В качестве пространства состояний $\mathcal{S}$ предлагается взять временную составляющую S_t, неконтролируемую составляющую – S_x и управляемую составляющую – S_c:

$$\mathcal{S} = S_t \times S_x \times S_c.$$

Временная составляющая S_t будет содержать информацию о состоянии энергосистемы, управляемая составляющая S_c отвечает за информацию о состоянии среды (текущий заряд батареи), а неуправляемая составляющая S_x будет использовать информацию из исторических данных, наблюдаемую в текущий момент и оказывающую влияние на динамику системы и функцию затрат.

Пространство действий ($\mathcal{A}$). На каждом временном шаге t агенту необходимо выполнить действие над аккумуляторной батареей: зарядить, разрядить или ничего не предпринимать. Возможные действия агента должны удовлетворять $a_t \in [P_d^{max}; P_c^{max}]$, где величины P_d^{max} и P_c^{max} отвечают за мощность накопителя в рамках 15 минут с учетом всех его индивидуальных характеристик. В рамках данной задачи пространство дей-

ствий может быть как непрерывным, так и дискретным, с некоторым заранее заданным числом действий.

Множество наград ($\mathcal{R}$). Целью агента в рамках данной задачи является минимизация среднего значения относительных затрат на покупку электроэнергии с использованием батареи к затратам на покупку электроэнергии без батареи:

$$score = \frac{M_b - M}{|M|},$$

где M_b затраты с батареей, M затраты без батареи.

Ограничение на емкость батареи. Обучение с подкреплением не позволяет явным образом определять ограничения задачи. Дополнительной и немаловажной целью в рамках данной проблемы является выбор действий, удовлетворяющих ограничению емкости аккмулуяторной батареи, чтобы текущий уровень энергии в аккумуляторной батареи рано или поздно не вышел за обозначенные границы. Решением чаще всего является добавление к функции награды некоторого штрафа (вознаграждения) в случае нарушения (соблюдения) необходимого условия.

Некорректное определение функции награды при наличии нескольких целей у агента может существенно замедлить процесс обучения, поэтому не рекомендуется использование экспоненциальных функций в качестве штрафов в методах типа ‘off-policy’, а также при наличии малой по модулю награды основной цели агента.

3.2 Ray

Параллельно с развитием области машинного обучения, возникает необходимость в использовании параллелизма. На создание инфраструктур, обладающих эффективным использованием, например, алгоритмов обучения с подкреплением, специалистам приходится тратить массу усилий и времени для подготовки процесса обучения модели на крупных машинах или кластерах. С этой целью в рамках данной работы предлагается использовать Ray, который позволяет пользователю не задумываться о планировании, передаче данных и машинных сбоях, но при этом обладает преимуществами оптимизированной вручную системы в смысле ее произ-

водительности.

Ray представляет из себя простой и универсальный интерфейс программирования приложений (API) для языков Python и Java [6], библиотеки и инструменты которого позволяют пользователям работать над самыми разнообразными задачами Machine Learning, начиная от распределенного обучения или подбора гиперпараметров модели, заканчивая Deep Reinforcement Learning и обслуживанием производственных моделей. Отличительными особенностями данной библиотеки от, например, не менее известной OpenAI Baselines, является распределенное управление памятью, низкоуровневый контроль над ресурсами вычислительного устройства. Воспользоваться этим интерфейсом возможно как на собственных

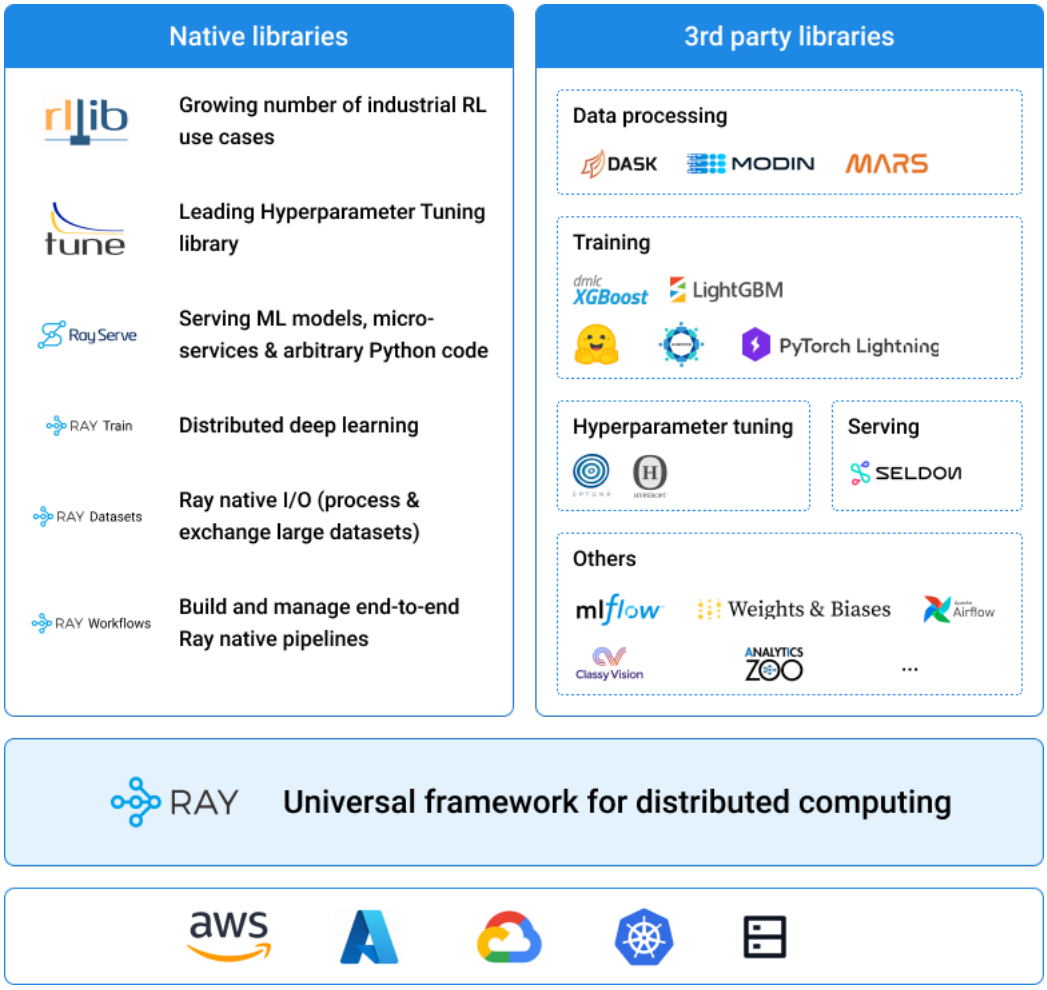


Рис. 14: Система Ray

серверах, так и на серверах популярных облачных провайдеров: Amazon

Web Services (AWS), Google Cloud Platform (GCP), Azure. Помимо этого, Ray продолжает интегрировать многие популярные библиотеки, которые базируются на обработке данных, обучении моделей, тюнинге гиперпараметров и многие другие. Обзор системы Ray представлен на следующей картинке:

Ray в силу своей гибкости, позволяет работать как над простыми проблемами машинного обучения, где необходимо простое применение алгоритма машинного обучения, так и над реальными индустриальными задачами, для решения которых возникает необходимость в написании собственных методов обработки данных, алгоритмов обучения и других его составляющих.

В рамках данной работы использовались 2 библиотеки расширения Ray Machine Learning - Ray Tune и Ray RLlib. Для создания среды обучения использовались инструменты интерфейса OpenAI Gym, позволяющие воспроизвести среду, подходящую для любого алгоритма Reinforcement Learning.

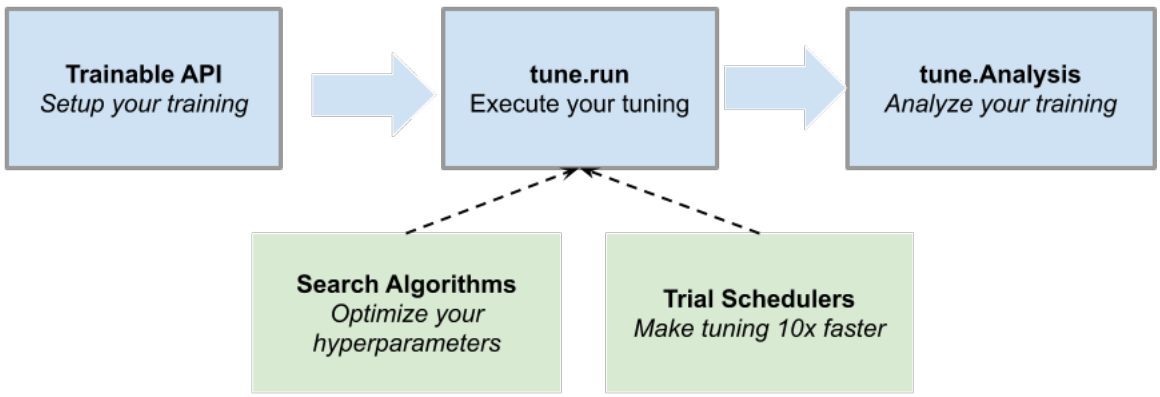


Рис. 15: Рабочий процесс Tune API

3.2.1 Ray Tune

Rtune (Ray Tune) – представляет собой модуль, отвечающий за методы оптимизации гиперпараметров (тюнинг). Основными средствами поиска и оптимизации гиперпараметров являются:

- алгоритмы оптимизации (Schedulers)

- алгоритмы поиска (Search Algorithms)

При работе с алгоритмами поиска гиперпараметров имеется возможность задавать как интервал их поиска, так и конечное множество их значений. Основным принципом работы scheduler-ов является отсечение заранее неудачных тренировочных сессий и выделение ресурсов под наиболее обещающие случаи (см. рис. 16). Наиболее популярными являются Population Based Training [31] и Asynchronous HyperBand [32].

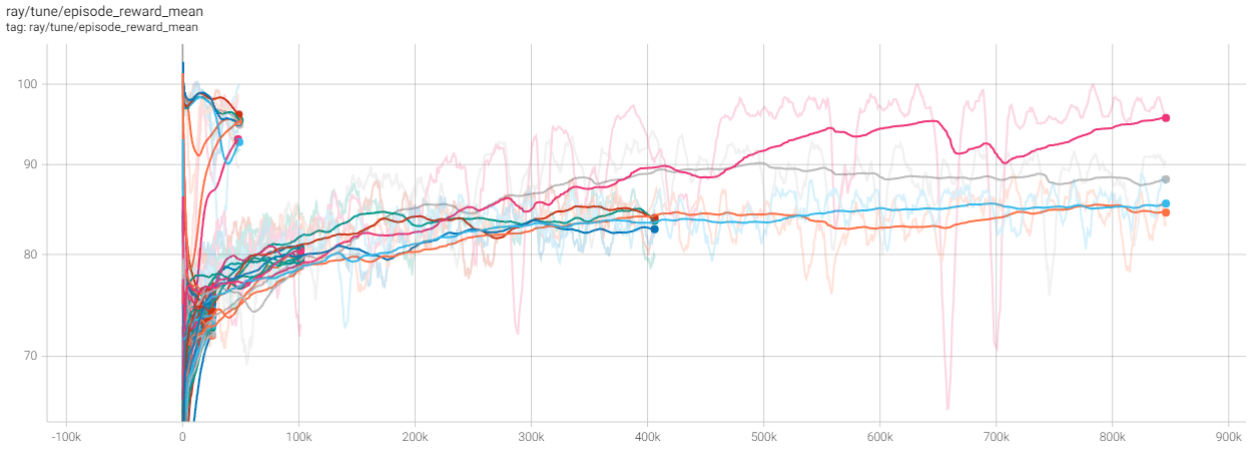


Рис. 16: Пример работы Hyperband scheduler

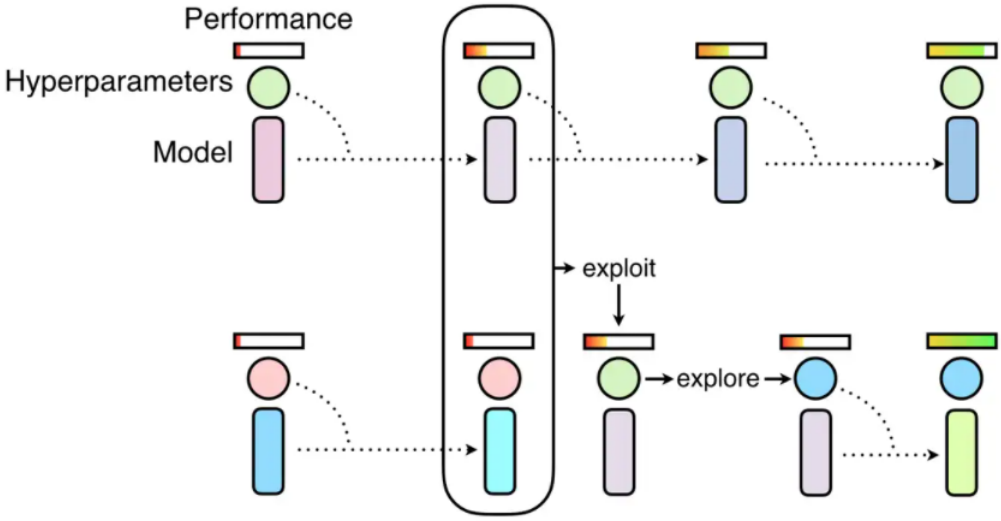


Рис. 17: Принцип работы Population Based

На рисунке 17 изображен пример работы Population Based Training, в ко-

тором из двум моделей выбирается та, что продемонстрировала более удовлетворительный результат, продолжая в дальнейшем работать именно с ней.

3.2.2 Ray RLlib

Ray RLlib – библиотека с открытым исходным кодом для Reinforcement Learning, имеющая полную совместимость с OpenAI Gym [33]. При достаточных навыках работы с данным модулем пользователь способен не только экономить время при обучении моделей, но и при необходимости создавать алгоритмы вручную, сохраняя при этом все упомянутые ранее преимущества Ray.

Рис. 18: Поддерживаемые RLlib алгоритмы

Отметим наиболее значимые возможности, позволяющие избежать необходимость использования навыков сложной программной инженерии:

- поддержка популярных фреймворков глубокого обучения (PyTorch и Tensorflow)
- высокораспределенное обучение (возможность параллельного обучение на сотнях CPU)
- поддержка векторизованных сред
- поддержка мультиагентного Reinforcement Learning
- имитационное обучение и клонирование поведения

3.3 Выбор алгоритмов обучения с подкреплением

На практике в общем случае выбор алгоритма обучения с подкреплением напрямую зависит от решаемой задачи и следующих факторов:

- Model-Free или Model-Based
- Off-policy или On-policy
- непрерывность пространства действий среды $\mathcal{A}$
- узкая специализированность алгоритма

В первую очередь необходимо понять, имеется ли реальная модель рассматриваемой среды, которая бы позволяла точно определить вероятности переходов между ее состояниями. Во-вторых, для обеспечения хорошей скорости обучения алгоритма необходимо правильно сопоставить тип метода с выбранной функцией награды, ведь тип off-policy не во всех случаях способен быстро обучаться при наличии пошаговой непрерывной функции награды. В-третьих, между возможностью алгоритма и непрерывностью пространства действий среды должна быть совместимость. И наконец, некоторые алгоритмы являются узкоспециализированными и подходят для работы лишь с определенными видами задач, например, обучение по изображением алгоритмом Dreamer.

Учитывая все упомянутые ранее факторы, можно получить список наиболее привлекательных для задачи алгоритмов (4):

Таблица 4: Алгоритмы RL

Алгоритм	Model	Policy	Action space	Специализ.
A3C	Model-Free	On-policy	Disc. + Cont.	Нет
DDPG	Model-Free	Off-policy	Cont.	Нет
PPO	Model-Free	On-policy	Disc. + Cont	Нет
PG	Model-Free	On-policy	Disc. + Cont	Нет
SAC	Model-Free	Off-policy	Disc. + Cont	Нет

В рамках данной работы предлагается воспользоваться только методами категории on-policy learning, поскольку в этом случае не нужно уделять большое количество времени на определение функции награды среды, которая бы соответствовала ранее упомянутым требованиям ускорения скорости обучения агента.

Глава 4. Реализация

В рамках данной работы основные компоненты среды определялись следующим образом:

- **Пространство состояний** $\mathcal{S} = S_t \times S_x \times S_c$.
  Временная составляющая S_t содержит информацию о дне недели $S_t^d \in R$ и четверти дня $S_t^q \in R$:

$$S_t = S_t^d \times S_t^q.$$

  Управляемая составляющая S_c отвечает за информацию о состоянии среды, определяется текущим уровнем заряда батареи $B \in R$.
  В качестве компонент неуправляемой составляющей S_x берется информацию из исторических данных, наблюдаемую в текущий момент и оказывающую влияние на динамику системы и функцию затрат:

$$S_x = S_x^{l_0} \times S_x^{pv_0} \times S_x^{b_0} \times S_x^{s_0},$$

  где $S_x^{l_0}, S_x^{pv_0}, S_x^{b_0}, S_x^{s_0} \in R$ – прогнозные значения для текущего шага жилой нагрузки, выработки энергии фотоэлектрической станции, тарифов на покупку и продажу энергии. Использование огромного числа прогнозных значений, содержащихся в исторических данных на каждом временном шаге было принятно не использовать, поскольку это существенно замедлит скорость обучения модели, а также не повлияет на итоговые результаты модели в силу принципа, по которому происходит обучение моделей в рамках обучения с подкреплением.

  В результате состояние системы энергоснабжения определяется как:

$$S = (S_t^d, S_t^q, B, S_x^{l_0}, S_x^{pv_0}, S_x^{b_0}, S_x^{s_0}) \in \mathcal{S}.$$

- **Пространство действий** $\mathcal{A}$. С целью получения более точных и оптимальных результатов выбирается непрерывное пространство действий, где любое действие $a_t \in [P_d^{max}; P_c^{max}]$, $\forall t$.

- **Функция вознаграждения $\mathcal{R}$**. Для того чтобы удовлетворить целям агента – ограничение на емкость и минимизация затрат, предлагается использовать пошаговую награду r_t за удовлетворение ограничения и терминальную награду r_e за минимизации затрат.

  Пошаговую награду агент получает на каждом временном шаге и определяется она следующим образом:

$$r_t = \frac{t_i}{(t_i + t_r)},$$

  где t_i это номер шага в текущем эпизоде (горизонт планирования равный 1 дню), а t_r это номер шага на котором последний раз агент нарушил ограничение емкости батареи. Задание вознаграждения таким образом позволит агенту преследовать цель удовлетворения ограничению, в отличие от пошаговых наград в виде констант, используя которые агент будет менее чувствительным к выполнению поставленной задачи.

  Терминальную награду агент получает в конце модериуемого эпизода (горизонт планирования), который изначально был задан как 1 день (96 шагов), и определяется она как:

$$r_e = -score * H(-score),$$

  где $H(\cdot)$ – это функция Хевисайда, равная 1 при $score \leq 0$ и 0 иначе.

4.1 Результаты

Для проведения этапа обучения модели был выбран облачный сервис Microsoft Azure с виртуальной машиной общего назначения ‘Standard DS3 v2’, обладающей следующими характеристиками:

- OS: Linux (дистрибутив Ubuntu)
- CPU: Intel Xeon E5-2673 v3 2.4 GHz (Haswell)
- RAM: 14 GiB (SSD)

Для сравнения результатов работы алгоритмов RL была взята ранее упомянутая метрика *score*, определяющая среднее значение относительных затрат на покупку энергии у местной коммунальной сети с использованием батареи к затратам на ее приобритение без накопителя:

$$score = -\frac{M_b - M}{|M|},$$

где M_b затраты с батареей, M затраты без батареи.

Ниже представлены результаты (см. таб. 5) работы алгоритмов R, где в качестве метрики на этапе обучения моделей была выбрана средняя награда за эпизод:

метод	score	сэкономлено (%)
PPO	0.071	7.1%
A3C	0.093	9.3%
PG	0.037	3.7%
MILP	0.16	16%

Таблица 5: Итоговые результаты работы алгоритмов RL в сравнении с MILP

На основе результатов стоит отметить, что несмотря на ограничение времени обучения агента и отсутствие более высоких вычислительных мощностей, алгоритмы RL показали хорошие результаты при решении данной задачи. Можно произвести увеличение скорости обучения агента за счет подбора гиперпараметров и/или корректировок в использовании различных метрик внутри самих алгоритмов, в таком случае методы обучения с подкреплением позволят продемонстрировать куда большей экономии в смысле итоговых финансовых затрат.

Исходя из таблицы результатов проведенных экспериментов алгоритмов RL – A3C показал себя наилучшим образом с точки зрения итогового результата и скорости его получения. Из визуализации процесса обучения (см. рис. 19) можно видеть, что метод A3C для исследования стратегий снижения затрат существенно выходил за рамки ограничения емкости аккумуляторной батареи (значение 0), в то время как PPO удавалось удовлетворять ему большую часть этапа обучения.

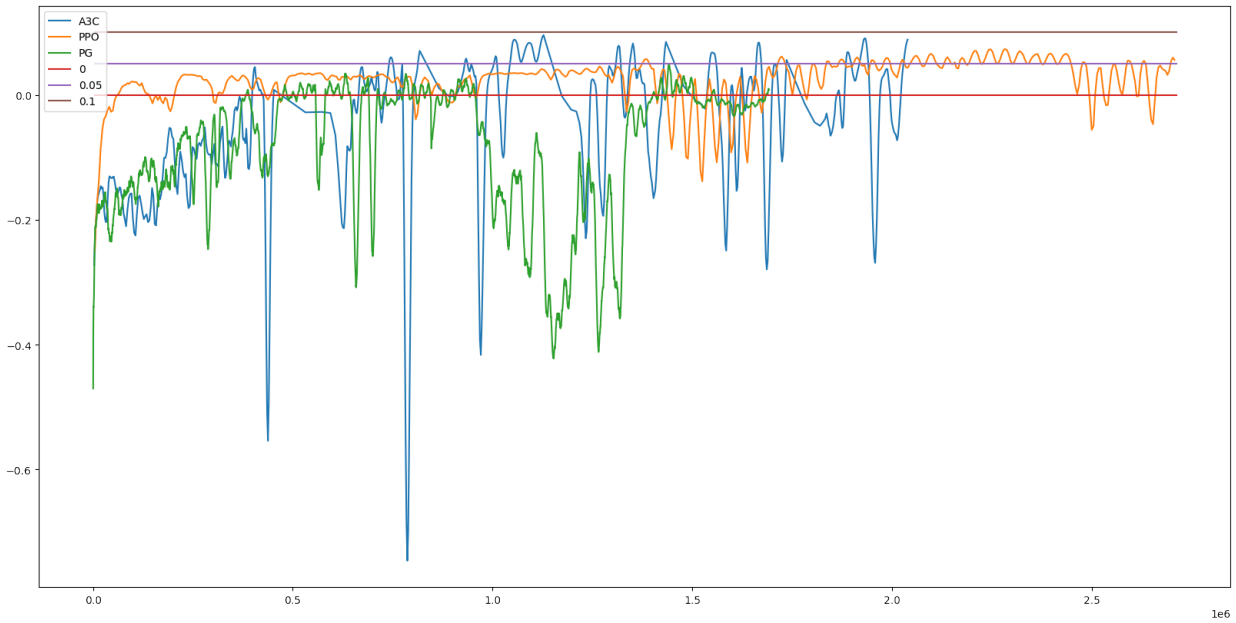


Рис. 19: График метрики score в процессе обучения моделей RL

Альтернативой при решении задачи может стать ее упрощение, сделав пространство действий $\mathcal{A}$ дискретным, что может поспособствовать скорости обучения агента. В дополнении следует использовать технику masking [34], работающую лишь со средой с дискретным пространством действий и позволяющую в несколько раз повысить эффективность обучения модели за счет отсечения действий, которые бы нарушали ограничения задачи. Это позволит избавиться от пошаговой награды, что позволит алгоритму сосредоточиться только на минимизации суммарных затрат.

Выводы

В задаче планирования использования батареи в энергосистеме с возобновляемыми источниками энергии и возможностью обмена с местной коммунальной сетью, подход смешанного целочисленного линейного программирования продемонстрировал ряд существенных недостатков. Прогнозные значения затрудняют использование модели при решении проблем долгосрочного планирования, не имеется возможности учитывания изменений во входных значениях, ввиду чего возникает потребность в повторном решении задачи с новыми данными, а отсутствие адаптивности подхода не позволяет применять его в рамках реальных индустриальных задач.

Автором работы предлагается воспользоваться не рассмотренным в рамках соревнования Power Laws подходом – методами обучения с подкреплением. Они позволяют работать с неопределенностями в прогнозных значениях, учитывать важные изменения во входных данных, например, эффект деградации батареи, а также в случаях возникновения незнакомых агенту ситуаций, адаптироваться под них, что делает этот подход очень эффективным в рамках применения к исследуемой задаче.

Заключение

В рамках работы были выполнены все поставленные задачи. Построена математическая модель рассматриваемой проблемы, проведен анализ исторических данных, исследованы и имплементированы алгоритмы обучения с подкреплением, а также сделано сравнение их результатов с подходом MILP, показавшего неплохие результаты в рамках соревнования Power Laws.

В результате исследований рассматриваемым алгоритмам обучения с подкреплением (PPO, A3C, PG), работающим с непрерывным пространством действий и относящимся к on-policy learning, удалось достичь хороших результатов. Увеличение времени обучения и повышение вычислительных мощностей позволит данным методам достичь еще лучших показателей с точки зрения экономии финансовых затрат, а также повысить скорость выполнения этапа обучения моделей.

Список литературы

[1] Официальный сайт schneider electric [Электронный ресурс] / SE. Режим доступа: `https://www.se.com/ru/ru/`, свободный. (дата обращения: 6.05.21)

[2] Репозиторий соревнования [Электронный ресурс] / GitHub. Режим доступа: `https://github.com/drivendataorg/power-laws-optimization`, свободный. (дата обращения: 6.05.21)

[3] Wang H., Huang T., Liao X. Reinforcement Learning for Constrained Energy Trading Games With Incomplete Information // IEEE Trans. Cybern, 2017. Vol. 47, P. 3404–3416.

[4] Kim B., Zhang Y. Dynamic Pricing and Energy Consumption Scheduling With Reinforcement Learning // IEEE Trans. Smart Grid, 2016. Vol. 7, P. 2187–2198.

[5] Ruelens F., Claessens B.J., Vandael S. Residential Demand Response of Thermostatically Controlled Loads Using Batch Reinforcement Learning // IEEE Trans. Smart Grid, 2017. Vol. 7, P. 2149–2159.

[6] Ray // 2022
URL: https://docs.ray.io/en/releases-1.1.0/index.html (дата обращения: 12.05.22)

[7] Dulout J., Hernandez L. Optimal Scheduling of a Battery-based Energy Storage System for a Microgrid with High Penetration of Renewable Sources // ELECTRIMACS Conference, 2017. P. 1–6.

[8] Chaouachi A., Rashad M., Kamel M. Multiobjective Intelligent Energy Management for a Microgrid // IEEE Transactions on Industrial Electronics, 2013. Vol. 60, No. 4, P. 1688–1699.

[9] Hatziargyriou N. Special issue on microgrids and energy management // Eur Trans Electr Power, 2011. Vol. 21, P. 1139–1141.

[10] Mohamed F.A., Koivo H.N. System modelling and online optimal management of MicroGrid with battery storage // International Journal on Electrical Power and Energy Systems, 2010. Vol. 32, No. 5, P. 398–407.

[11] Atia R., Yamada N. Sizing and analysis of renewable energy and battery systems in residential microgrids // IEEE Transactions on Smart Grid, 2016. Vol. 7, No. 3, P. 1204–1213.

[12] Bahramirad S., Reder W., Khodaei A. Reliability-constrained optimal sizing of energy storage system in a microgrid // IEEE Transactions on Smart Grid, 2012. Vol. 3, No. 4, P. 2056–2062.

[13] Gengo T., Kobayashi Y. Development of Grid-stabilization Power-storage System with Lithium-ion Secondary Battery // Mitsubishi Heavy Industries Technical Review, 2009. Vol. 46, No. 2, P. 36–42.

[14] Perez A., Moreno R. Effect of Battery Degradation on Multi-Service Portfolios of Energy Storage // IEEE Transactions on Sustainable Energy, 2016. Vol. 7, P. 1718–1729.

[15] Ruelens F., Claessens B.J., Vandael S. Residential Demand Response of Thermostatically Controlled Loads Using Batch Reinforcement Learning // IEEE Trans. Smart Grid, 2017. Vol. 7, P. 2149–2159.

[16] Xiong R., Cao J., Yu Q. Reinforcement learning-based real-time power management for hybrid energy storage system in the plugin hybrid electric vehicle // Appl. Energy, 2018. Vol. 211, P. 538–548.

[17] Kim B., Zhang Y. Dynamic Pricing and Energy Consumption Scheduling With Reinforcement Learning // IEEE Trans. Smart Grid, 2016. Vol. 7, P. 2187–2198.

[18] Sutton R.S., Barto A.G. // Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 2018.

[19] Wei C., Zhang Z., Qia W. Reinforcement learning based intelligent maximum power point tracking control for wind energy conversion systems // IEEE Trans. Ind. Electron, 2015. Vol. 62, No. 10, P. 6360–6370.

[20] Xi L., Yu L., Fu Y. Automatic generation control based on deep reinforcement learning with exploration awareness // Proc. CSEE, 2019. Vol. 39, No. 14, P. 4150–4162.

[21] Wang B., Zhou M., Xin B. Analysis of operation cost and wind curtailment using multi-objective unit commitment with battery energy storage // Energy, 2019. Vol. 178, P. 101–114.

[22] Wan Z., Li H., He H. Model-Free Real-Time EV Charging Scheduling Based on Deep Reinforcement Learning // IEEE Transactions on Smart Grid, 2019. Vol. 10, No. 5, P. 5246–5257.

[23] Sutton R.S., McAllester D., Singh S., Mansour Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation // Advances in Neural Information Processing Systems, 2000, Vol. 12, P. 1057–1063

[24] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. // Proximal Policy Optimization Algorithms, ArXiv, 2017, abs/1707.06347

[25] V. Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, A. Graves, T. Lillicrap, Tim Harley, D. Silver and K. Kavukcuoglu. // Asynchronous Methods for Deep Reinforcement Learning, ArXiv, 2016, abs/1602.01783

[26] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra // Continuous control with deep reinforcement learning, ArXiv, 2015, abs/1509.02971

[27] Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning // Machine Learning, 1992. Vol. 8, P. 229–256.

[28] Hunter D. R., Lange K. A Tutorial on MM Algorithms // The American Statistician, 2004. Vol. 58, P. 30–37.

[29] Wu, C. F. Jeff On the Convergence Properties of the EM Algorithm // Annals of Statistics, 1983. Vol. 11, P. 95–103.

[30] Schulman J., Levine S., Moritz P., Jordan M. I., Abbeel P. // Trust region policy optimization, ArXiv, 2015, abs/1502.05477

[31] Population Based Training // 2017
URL: https://www.deepmind.com/blog/population-based-training-of-neural-networks (дата обращения 12.05.22)

[32] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar // Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization, ArXiv, 2018, abs/1603.06560

[33] OpenAI Gym // 2022
URL: https://www.gymlibrary.ml/l (дата обращения: 12.05.22)

[34] Huang S., Ontanon S. // A Closer Look at Invalid Action Masking in Policy Gradient Algorithms, ArXiv, 2020, abs/2006.14171