МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

ПЕЧЕНЕВ Николай Леонидович

ИССЛЕДОВАНИЕ МЕТОДОВ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ НА ПРИМЕРЕ КОМПЬЮТЕРНЫХ ИГР

Магистерская диссертация
специальность 1-31 80 09 «Прикладная математика и информатика»

Научный руководитель
Марков Сергей Викторович,
кандидат физико-математических
наук, доцент

Допущена к защите
«31» марта 2021 г.
Заведующий кафедрой дискретной математики и алгоритмики
Котов Владимир Михайлович
доктор физико-математических наук, профессор

Минск, 2021

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ 3
ВВЕДЕНИЕ 4
ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ 6
ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ 9
1.1 Среда 9
1.2 Наблюдение 9
1.3 Типы действий 9
1.4 Функция награды 10
ГЛАВА 2 АРХИТЕКТУРА СЕТИ 10
2.1 Модули 10
2.2 Полная модель 11
2.3 Модель с усеченным набором действий без выбора параметров 12
2.4 Модель с усеченным набором действий без выбора параметров и обучением на статистиках 14
ГЛАВА 3 ЭКСПЛОРИНГ 15
3.1 Дерево зависимостей 15
3.2 Резервирование 16
3.3 Оценка качества эксполринга 17
ГЛАВА 4 ФУНКЦИЯ НАГРАДЫ 18
4.1 Проблема внутренней функции награды 18
4.2 Счет как функция награды 18
4.3 Распределение награды по дереву зависимостей 20
4.4 Обобщенная оценка преимущества 23
ГЛАВА 5 АЛГОРИТМЫ ОСНОВАННЫЕ НА ПОЛИТИКЕ 25
5.1 Алгоритм подкрепления 25
5.2 Актор-Критик с преимуществами 27
5.3 Проксимальная оптимизация политики 30
5.4 Проксимальная политика оптимизации с советчиком 33
5.5 Феодальная модель для иерархического обучения с подкреплением 37
ГЛАВА 6 ПРОЦЕСС ОБУЧЕНИЯ 39
ЗАКЛЮЧЕНИЕ 44
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 46

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

1. Юнит — боевая единица, или рабочая единица, или здание, или ресурс
2. Актор — действующее лицо
3. Агент — тот, кто может взаимодействовать со средой посредством действий
4. Эксполринг — процесс исследования среды
5. Преимущества — полученная награда за вычетом оценки состояния
6. Раннер — объект, который запускает среду и налаживает взаимодействие агента со средой
7. Сэмплер — объект, который генерирует подвыборки из траектории для тренировки
8. CPU - central processing unit, центральный процессор
9. GPU - graphics processing unit, графический процессор
10. RNN – рекуррентная нейронная сеть
11. CNN – свёрточная нейронная сеть
12. Starcraft II – компьютерная игра, среда для обучения агента, жанр игры – стратегия в реальном времени
13. Python – высокоуровневый язык программирования
14. Протосс – фракция в Starcraft II

ВВЕДЕНИЕ

В ходе совершенствования вычислительных машин, активно развиваются информационные технологии, в том числе разработки в области обучения с подкреплением, которые за последнее время получили большой толчок[1]. Множество крупных компаний активно развиваются в этой сфере.

Стоит следующая задача: построить искусственный интеллект, далее будет называться агентом, для игры StarCraft II используя малые вычислительные мощности.

Данная задача принадлежит следующему классу: задача с мультиагентной средой, континуальным пространством состояний, континуальным пространством действий, неполным наблюдением, различными начальными состояниями и разреженной функцией награды. Если говорить простым языком, то в данной игре принимают участие больше одного игрока. От каждого игрока частично скрыта карта туманом войны, и каждый из игроков может наблюдать практически бесконечное множество вариантов событий и в каждый момент может выбрать действие из практически бесконечного множества. Начинать игру они могут на разных картах, и результат подводится только в конце партии без промежуточных результатов.

Из недостатков текущих решений стоит отметить, что некоторые исследования проводятся в специальных матчах, где среда сильно изменена. Например, в ней может присутствовать только тактическая составляющая на управление войсками, а экономика, технологии и производство отсутствуют[2]. Некоторые используют оригинальную среду, но экономику, производство и технологии передают в распоряжение детерминированному алгоритму, оставляя на обучение только тактику. Другие, используя оригинальную среду, изменяют текущий набор действий на спроектированные комбинации действий, которые они называют макродействиями, и изменяют пространство наград, но обучение и тестирование проводят в крайне необъективных условиях[3]. Но больше всего статей завязаны на использовании записей игр профессиональных игроков. Тот же AlphaStar компании DeepMind сначала обучался на записях игр до очень высокого уровня, и только после этого включалось обучение с подкреплением, которое все еще использовало опыт записанных игр.

Сложность такой задачи заключается в следующем: из-за наличия нескольких агентов, обучаемый агент должен уметь подстраиваться под действия оппонента, а не делать некую последовательность действий, рассчитывая на неизменяемость среды. Континуальное пространство состояний говорит, что состояния нужно уметь кодировать, причем похожие состояния должны похоже кодироваться. Континуальное пространство действий говорит о том, что в общем случае нельзя предсказывать действия в виде дискретного распределения. Неполные наблюдения тоже нужно учитывать, особенно важно сохранять информацию в памяти. Разные начальные состояния мешают тем, что нельзя переобучаться на одной карте, а проблема разреженности функции награды заключается в том, что сложно связать награду с действием, которое ее дало. В ходе работы будут реализованы модели разной степени сложности, но большая часть экспериментов пройдет на упрощенной модели без континуального пространства действий и с постоянным начальным состоянием.

Для простоты была выбрана одна из трех фракций – протоссы. Обучаться бот будет именно на этой фракций, причем для того, чтобы у него был вариант играть самим с собой, тренироваться он будет тоже против фракции протосс.

Программа пишется на языке Python. Для взаимодействия со средой используется библиотека pysc2. Для построения нейронных сетей используется библиотека pytorch.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Магистерская диссертация, 46 страницы, 31 рисунков, 10 источников.

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, АГЕНТ, СРЕДА, STARCRAFT.

Объект исследования: алгоритмы обучения с подкреплением, нейросетевые модели.

Цель работы: изучить методы обучения с подкреплением, разработать и реализовать алгоритмы обучения нейросетевых моделей, провести сравнительный анализ полученных результатов, определить возможные направления работы для улучшения качества работы алгоритмов.

Методы исследования: анализ, эксперимент, тестирование, сравнение.

В ходе работы были выявлены особенности среды, которые не позволяли обучать агента, а именно: проблемы эксплоринга и разреженной функции награды. Для каждой задачи были разработаны методы их решения. Далее был проведен обзор наиболее распространенных нейросетевых методов обучения, после чего они были применены для обучения агента. Был проведен сравнительный анализ результатов, по итогом которого были предложены возможные направления дальнейших исследований.

Результат работы: исследовано состояние проблематики задачи, изучена литература по методам обучения с подкреплением, решены проблемы исследования среды, смоделирована функция награды, построен, обучен и протестирован нейросетевой агент.

Область применения – разработка искусственного интеллекта, робототехника, обработка естественного языка.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Магістарская дысертацыя, 46 с., 31 мал., 10 крыніц.

НАВУЧАННЕ З ПАДМАЦАВАННЕМ, МАШЫННАЕ НАВУЧАННЕ, НЕЙРОНАВЫЯ СЕТКІ, АГЕНТ, СЕРАДА, STARCRAFT.

Аб'ект даследавання: алгарытмы навучання з падмацаваньнем, нейросетевая мадэлі.

Мэта працы: вывучыць метады навучання з падмацаваньнем, распрацаваць і рэалізаваць алгарытмы навучання нейросетевая мадэляў, правесці параўнальны аналіз атрыманых вынікаў, вызначыць магчымыя напрамкі работы для паляпшэння якасці працы алгарытмаў.

Метады даследавання: аналіз, эксперымент, тэставанне, параўнанне.

У ходзе працы былі выяўленыя асаблівасці асяроддзя, якія не дазвалялі навучаць агента, а менавіта: праблемы эксплоринга і разрэджанай функцыі ўзнагароды. Для кожнай задачы былі распрацаваны метады іх рашэння. Далей быў праведзены агляд найбольш распаўсюджаных нейросетевая метадаў навучання, пасля чаго яны былі ўжытыя для навучання агента. Быў праведзены параўнальны аналіз вынікаў, па вынікам якога былі прапанаваны магчымыя напрамкі далейшых даследаванняў.

Вынік працы: даследавана стан праблематыкі задачы, вывучана літаратура па метадах навучання з падмацаваньнем, вырашаны праблемы даследавання асяроддзя, змадэляваныя функцыя ўзнагароды, пабудаваны, навучаны і пратэставаны нейросетевая агент.

Вобласць прымянення: распрацоўка штучнага інтэлекту, робататэхніка, апрацоўка натуральнай мовы.

ABSTRACT

Master's dissertation, 46 pages, 31 pictures, 10 sources.

REINFORCED LEARNING, MACHINE LEARNING, NEURAL NETWORKS, AGENT, ENVIRONMENT, STARCRAFT.

The object of research: reinforcement learning algorithms, neural network models..

The aim of this work is to study reinforcement learning methods, develop and implement algorithms for training neural network models, conduct a comparative analysis of the results obtained, identify possible areas of work to improve the quality of the algorithms.

Research methods: analysis, experiment, testing, comparing.

During the course of the research the features of the environment were revealed that did not allow training the agent, namely, the problems of exploration and the sparse function of the reward. Methods for solving them were developed for each task. Next, we reviewed the most common neural network training methods, after which they were applied to train the agent. A comparative analysis of the results was carried out, as a result of which possible directions for further research were proposed.

The result of the work is a set of models built in the course of solving the task of training the agent in the environment. Separately, it is necessary to highlight the following algorithm: to solve the problems of exploration, one action can be replaced by another, if the requirements for the first are not met; added two additional reward functions, and each reward function has its own critic; The proximal policy optimization with an advisor was chosen as the main algorithm. As a result, an agent was obtained that competes on equal terms with the average built-in artificial intelligence.

Field of application: artificial intelligence development, robotics, natural language processing.

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ

1.1 Среда

Каждая раса имеет свое технологическое дерево. Матчи между соперниками проводятся на карте. Цель – уничтожить все здания соперника.

На карте расположены: ресурсы (минералы, газ), плато разной высоты, переходы между плато, воздушные зоны. Матч начинается с того, что каждому игроку дается главная база и 12 рабочих.

Суть: собирать ресурсы, улучшать экономику, создавать производство, нанимать войска, развивать технологии, принимать грамотные сражения.

1.2 Наблюдение

Каждый момент времени среда генерирует наблюдение для каждого из игроков с учетом тумана войны. Наблюдение состоит из: последовательности статистик для каждого видимого юнита на карте, экран, миникарта, общая статистика вроде количества ресурсов, лимит армии и т.д.

1.3 Типы действий

Действия бывают следующих типов:

1) Без действующего лица: глобальные действия, не требующих юнитов-акторов. Пример: выбрать группу, переместить экран.
2) Быстрое: действие, которое совершают юниты-акторы без дополнительных параметров. Пример: большая часть производства.
3) Точечное: действие, которое совершают юниты-акторы с указанием координаты на карте. Пример: атака по координате.
4) Целевое: действие, которое совершают юниты-акторы, с указанием юнита-цели. Пример: добыча рабочим друзы минералов.

Для каждого из этих действий нужно генерировать свой набор параметров.

1.4 Функция награды

В момент уничтожения всех своих зданий игрок получает награду -1. В момент уничтожения всех вражеских задний игрок получает 1. Во все остальные моменты награда равна 0. Ограничение по времени: 21 минута.25 секунд, либо 2881 шага, если за это время матч не закончен, то дается награда 0.

ГЛАВА 2 АРХИТЕКТУРА СЕТИ

В идеале предполагается, что сеть будет генерировать все параметры действия. Но обучение таких сетей будет занимать огромное количество времени. Учитывая, что простой запуск обучения, без тонны улучшений, заканчивается полным провалом, то разумным будет сделать легкую сеть и использовать ее для экспериментов.

2.1 Модули

Пусть модель состоит из модулей.

Кодирующий модуль переводит наблюдение O_i и память с предыдущего шага M_{i-1} в новую память M_i. Используется для получения действия или стоимости текущего состояния.

Наблюдение можно брать как целиком, так и брать по нему какую-то статистику. Если брать некую статистику по наблюдению, то нет проблем, чтобы эту информацию передает в нейронную сеть. Если брать наблюдение целиком, нужно решить проблему с тем, что там содержится информация разного рода, а именно статистика, последовательности, изображения. Статистику передаем в нейронную сеть через полносвязные слои, последовательности — через реккурентные нейронные сети и механизмы внимания, изображения — через сверточные нейронные сети.

Расширенный кодирующий модуль переводит наблюдение O_i, действие a_i и память с предыдущего шага M_{i-1} в новую память M_i. Нужен для получения параметров при известном действии в полной модели. По сути это как обычный кодирующим модуль, но только у него есть дополнительный вход — текущее действие.

Оценивающий модуль переводит память M_i в стоимость состояния V_i. Используется для критика в модели актор-критик с преимуществом.

Модуль действия предназначен для получения из памяти M_i вероятностей для категориального распределения действия a_i.

Модуль выбора действующего лица переводит M_i и текущий набор подконтрольных юнитов $PlayerU_i$ в вероятности быть действующим лицом текущего действия a_i. Это распределение Бернулли для каждого подконтрольного юнита. Используется в быстрых, точечных и целевых действиях.

Модуль выбора координаты переводит M_i и карту в вероятности координат на карте. Используется в точечных действиях. Распределение зависит от выбранной модели. Последняя версия использует категориальное распределение по пикселям карты[4].

Модуль выбора цели переводит M_i и последовательность всех юнитов $AllU_i$ в вероятности быть целью текущего действия a_i. Это категориальное распределение для всех юнитов. Используется в целевых действиях.

2.2 Полная модель

В случае, когда модель генерирует и действие a, и параметры для действия p, с наблюдением O, памятью M и параметрами сети θ ее модель можно описать следующим образом:

$$P(a,p|O,M,\theta) = P(a|O,M,\theta_1)P(p|O,M,a,\theta_2)\#(2.1)$$

Содержит в себе кодирующий модуль, расширенный кодирующий модуль, два оценивающих модуля, модуль действия, модуль выбора действующего лица, модуль выбора координаты, модуль выбора цели. Граф модели изображен на рисунке 2.1

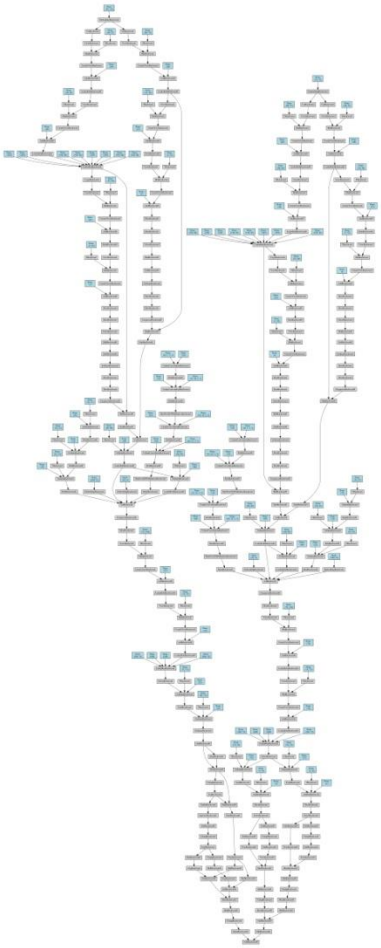

Рисунок 2.1 – Граф полной модели.

Очень тяжелая модель для обучения, так как использует наблюдения целиком и генерирует все параметры для действия. Обучается очень долго, что стало причиной использования более простых моделей.

2.3 Модель с усеченным набором действий без выбора параметров

Пусть параметры для действия a будут выбираться по неким эвристикам. Тогда при наблюдении O, имея память M и параметры сети θ:

$$P(a, p|O, M, \theta) = P(a|O, M, \theta)P(p|O, M, a)\#(2.2)$$

Схема того, как выглядит архитектура такой модели, изображена на рисунке 2.2.

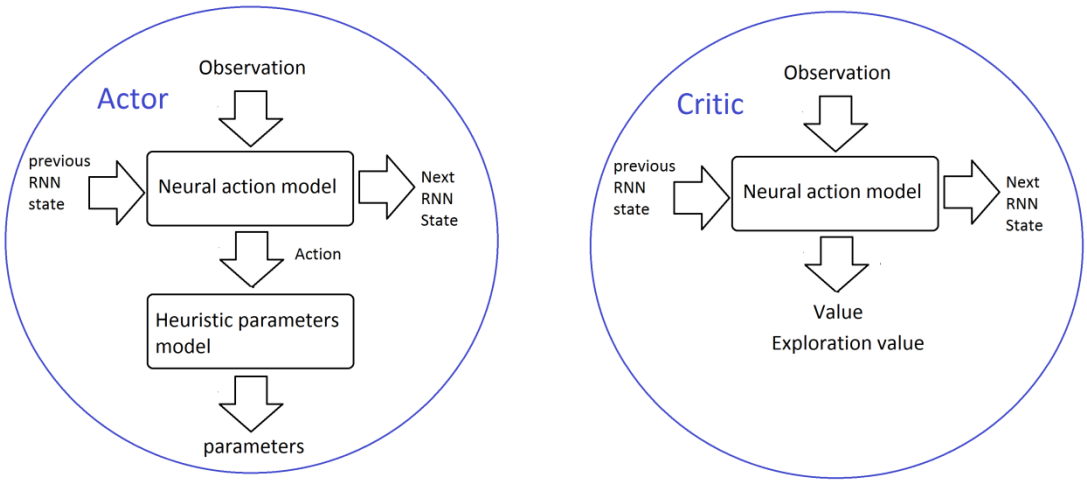

Рисунок 2.2 – Схема упрощенной модели.

В свою очередь модель выбора действия и модель получения оценки состояния в усеченной модели можно изобразить в качестве графа, как показано на рисунке 2.3.

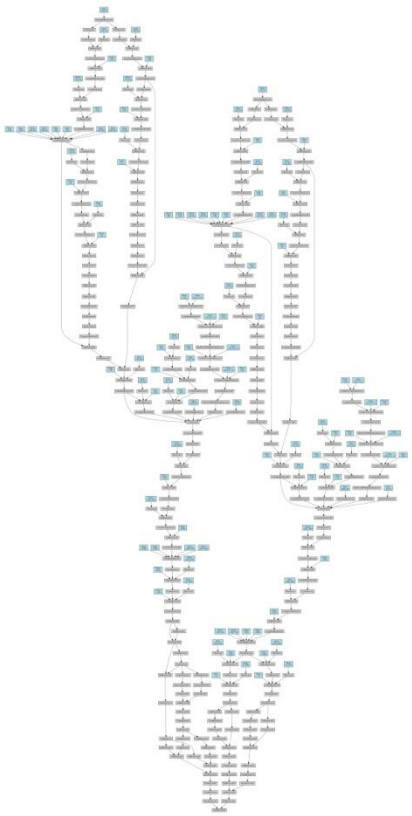

Рисунок 2.3 – Граф модели с усеченным набором действий без выбора параметров.

В результате итерации внутри среды стали быстрее в среднем на 50%. Итерации во время обучения - быстрее в 5 раз.

2.4 Модель с усеченным набором действий без выбора параметров и обучением на статистиках

Более того, в качестве наблюдений можно использовать некоторые статистики, посчитанные из O. Кроме того, можно сократить пространство действий. Граф модели уменьшается радикально, что можно проследить по рисунку 2.4.

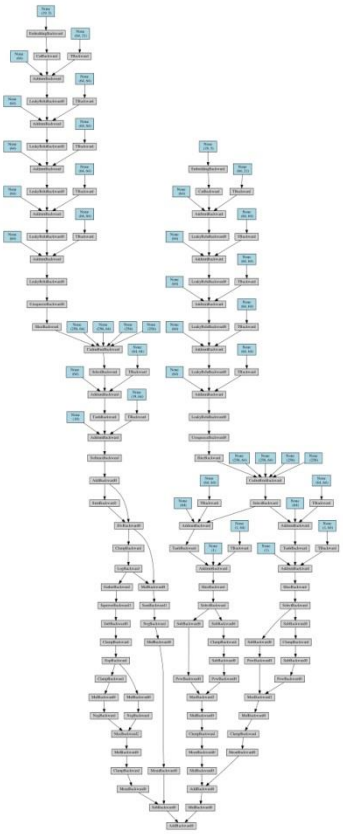

Рисунок 2.4 – Граф модели с усеченным набором действий без выбора параметров и обучением на статистиках

В результате итерации внутри среды стали быстрее в среднем на 60%. Итерации во время обучения - быстрее в 20 раз.

ГЛАВА 3 ЭКСПЛОРИНГ

3.1 Дерево зависимостей

Есть проблема, которая заключается в том, что некоторые действия расположены вверх по технологическому дереву, как показано на рисунке 3.1. Чтобы совершить такое действие, сначала нужно пройтись вверх по дереву, соответственно успешно совершив определенный набор действий.

Рисунок 3.1 – Технологическое дерево.

Например, для действия «Построить боевую единицу Сталкер» необходимо производственное здание «Врата» и технологическое здание «Кибернетическое ядро». Если дать среде на выполнение действие без выполненных условий, то среда не выполнит ничего.

Усугубляет проблему то, что частичный проход вверх по дереву может не сулить награды, в то время как полный проход может быть очень выгоден. Мы можем использовать маску, чтобы обнулить вероятности действий, для которых не выполнены технологические условия. Но в этом случае маленькая вероятность некоторых действий может обрубить целую технологическую ветку. То, как выглядит статистика действий на первой итерации, если не решать проблему эксплоринга, изображено на рисунке 3.2. Из показанной статистики видно, что агент выполняет только узкий набор действий, которые доступны только в начале игры.

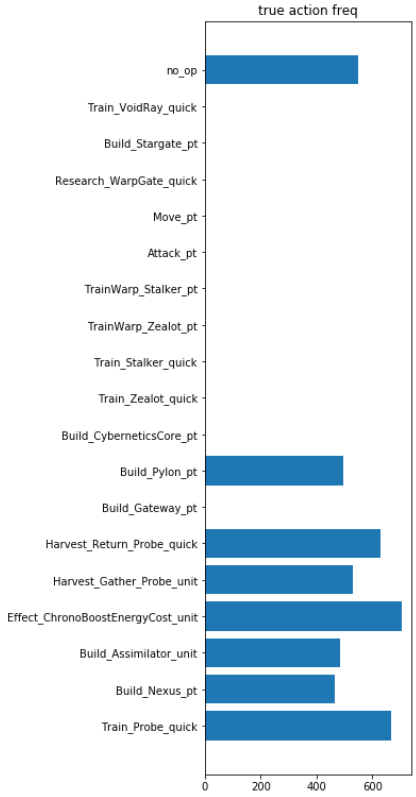

Рисунок 3.2 – Статистика действий в случае игнорирования проблем эксплоринга.

Предложение - разрешить действия, для которых не выполнены условия, и вместо них рекурсивно выполнять технологические требования для этого действия.

3.2 Резервирование

В связи с тем, что некоторые действия (такие как постройка зданий, найм боевых и рабочих единиц и т.д.) требуют ресурсы, происходит неприятная ситуация: необученный агент быстро перебирает все действия, и, при появлении ресурсов на самое дешевое действие, он тратит ресурсы на него. Таким образом у него не получается совершить затратное по ресурсам действие, хотя оно может быть достаточно выгодным. Часто агент не способен выбраться из этой ловушки.

Предложение - при выборе действия, на которое не хватает ресурсов, запасы временно замораживаются.

3.3 Оценка качества эксполринга

Для оценки качества эксплоринга среды сделаем следующее: для каждого варианта запускается необученный агент против самого слабого бота. Будем оценивать то, насколько хорошо агенты смогут покрыть технологическое дерево.

1) По умолчанию: если условия для действия не выполнены, то агент пропускает действие. В этом случае будет 70 % покрытие дерево суммарно за 5 игр и 55% покрытие дерева в среднем за каждую из игр.

2) Если условия для действия не выполнены, то вероятность выбора этого действия равна 0. В этом случае будет 50 % покрытие дерево суммарно за 5 игр и 41% покрытие дерева в среднем за каждую из игр.

3) Использовать дерево зависимостей и резервирование. В этом случае будет 82 % покрытие дерево суммарно за 5 игр и 64% покрытие дерева в среднем за каждую из игр.

В первом случае агент пропускает много действий, из-за чего накапливается много ресурсов. С одной стороны это позволяет выполнять дорогостоящие действия, с другой стороны большую часть игры агент бездействует, что не позволяет ему обучаться против более-менее сильных ботов.

Во втором случае агент делает много дешевых действий, и он плохо справляется с накоплением ресурсов для дорого действия.

ГЛАВА 4 ФУНКЦИЯ НАГРАДЫ

4.1 Проблема внутренней функции награды

Внутренняя функция награды это +1, 0, -1 в конце траектории за соответственно победу, ничью, поражение.

Награда очень разреженная и очень плохо отображает полезность действий. Учитывая сложность среды, обучиться на такой награде практически невозможно, что было показанное компанией DeepMind еще в 2017 году[5]. Это подтверждают и собственные эксперименты. На рисунке 4.1 изображен график, где только ничьи и поражения.

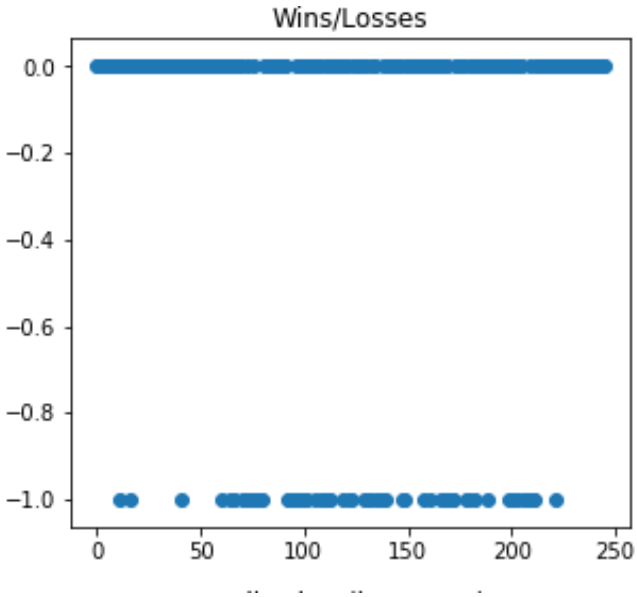

Рисунок 4.1 – Обучение агента по внутренней функции наград.

В этом случае награду либо заменяют, либо смешивают со собственноручно сделанной наградой. Для своей награды можно использовать внутренний счет игры.

4.2 Счет как функция награды

Пусть награда — это разница в счете между соседними промежутками времени.

Тогда в качестве награды можно брать взвешенную сумму счета. Эмпирически были выведены следующие правила: использованный лимит -

плохая статистика, сложно конвертировать в награду. Количество минералов и газа, потерянных противником в сражении - полезная статика, которая свидетельствует об успехе боевых действий, если рассматривать вместе со своим количеством потерянных минералов и газа. Использовано минералов и газа - не очень полезная статистика. При правильной игре ресурсы тратятся по максимуму, но заставлять это максимизировать напрямую приводит к плохим последствиям. Время простоя – полезная статистика для того, чтобы не строить лишние производящие здания и лишних рабочих, но может привести к недостатку рабочих и зданий. Собранное количество минералов и газа – относительно полезная статистика, прямая максимизация не приводит к победе, а иногда бывает очень вредной.

В случае долгого подбора коэффициентов и оптимизации гиперпараметров можно обучить агента со следующей статистикой против ботов.

AI	lose	draw	win
very_easy	0	5	5
easy	5	3	2
medium	9	1	0
medium_hard	10	0	0

Рисунок 4.2 – Статистика агента, обученного по внутренней функции наград.

Как показано на графике, даже после многочисленных побед, модель все равно скатилась в поражения и ничьи, что показано на рисунке 4.3. Награду нужно спроектировать таким образом, чтобы она была с одной стороны не сильно разряженной и адекватно награждала за правильные действия, но с другой стороны награда при любых обстоятельствах за победу должна давать больше, чем за ничью, а за ничью – больше, чем за победу. Даже в случае, если при ничье было произведено много армии и добыто много ресурсов, награда за победу должна быть все равно больше.

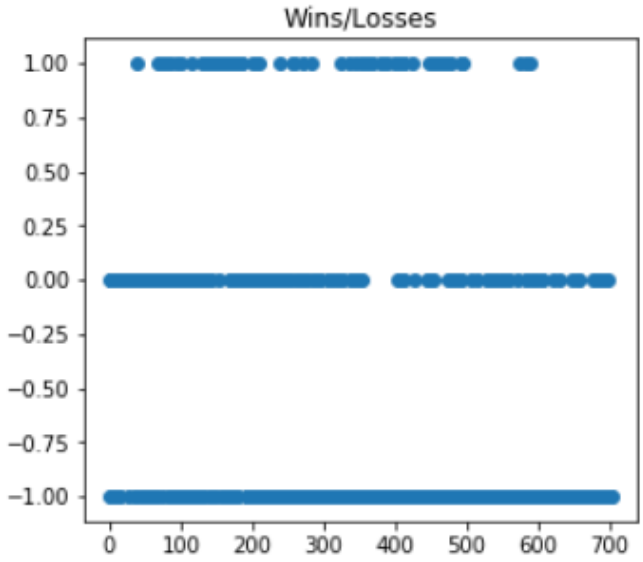

Рисунок 4.3 – Обучение агента с наградой как взвешенная сумма счета.

4.3 Распределение награды по дереву зависимостей

Для решения данной проблемы было необходимо экспериментировать с распределением функции награды.

Один из неудачных опытов – это попытка предсказания распределения награды. Это могло помочь с проблемой того, что у разных действий награда распределена очень по-разному. Например, боевой юнит получает награду за свое производство и уничтожения противников. В то время как награда рабочего за добычу ресурсов может быть размазана чуть ли не на всю игру.

Так как на эксперименты много времени не было, успех гарантированным не был, и исследований по данной теме я не нашел, то было задвинуть идею.

Были другие неудачные опыты с попыткой по некоторым эвристикам распределить награды. Например, распределять награды с учетом затраченных ресурсов.

Формализуем условие победы: побеждает игрок, у которого есть здания, а у противников их нет. Формулу можно переписать следующим образом:

$$|MyBuildings| > 0 \;\&\; |EnemyBuildings| == 0 \Rightarrow$$
$$MyBuildingsValue > 0 \;\&\; EnemyBuildingsValue == 0 \quad \#(4.1)$$

Преобразуем формулу в то, что можно максимизировать. Вторую часть формулы можно переделать следующим образом:

$$EnemyBuildingsValue == 0 => KilledEnemyBuildings \rightarrow \max \#(4.2)$$

Первую же часть формулу так просто не преобразуешь. Напрямую максимизировать суммарную стоимость зданий — плохая идея. Но по опыту можно сказать, что в случае, если армия игрока больше армии его врага, то, скорее всего, врагу не удастся разрушить здания.

$$MyBuildingsValue > 0 => MyArmyValue > EnemyArmyValue =>$$
$$MyArmyProduced - MyArmyLost + KilledEnemyArmy \rightarrow \max \#(4.3)$$

В итоге:

$$MyBuildingsValue > 0 \ \& \ EnemyBuildingsValue == 0 =>$$
$$MyArmyProduced - MyArmyLost + KilledEnemyArmy$$
$$+ KilledEnemyBuildings \rightarrow max \ \#(4.5)$$

$$MyArmyProduced - MyArmyLost + KilledEnemy \rightarrow max \ \#(4.5)$$

Получается достаточно логичная формула. С большой армией будет меньше потерь и больше уничтоженных врагов, а для получения большой армии понадобится много ресурсов и много производящих зданий. Но как передать награду за производство, уничтожение врагов и штраф за потерю армии всей технологической, экономической и производственной цепочке?

Пусть для каждого юнита известны начало и конец его жизни. Награда за уничтоженный вражеский юнит распределяется по всем боевым юнитам с учетом их стоимости, если они живы в этом момент.

Построим граф зависимостей для каждого боевого юнита. Для постройки боевого юнита могут понадобиться: производящее задние, технологическое здание, минералы, газ, лимит. Для производящего здания и технологического здания тоже нужны ресурсы, и у них тоже могут быть технологические требования. Пример такого графа для Авианосца изображен на рисунке 4.4.

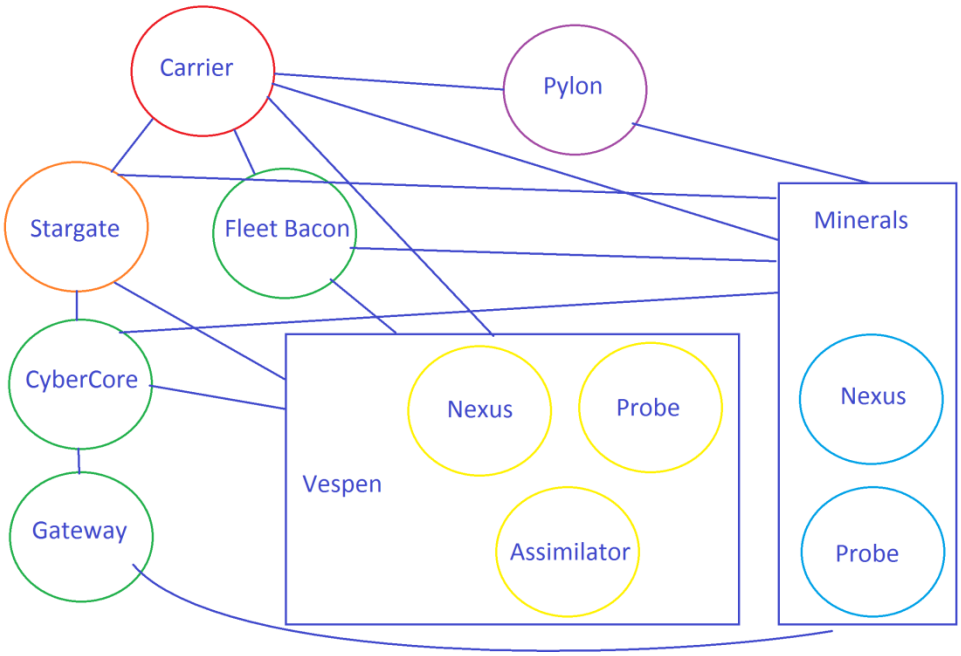

Рисунок 4.4 – Граф зависимостей для авианосца.

Таким образом, для каждого боевого юнита можно построить дерево, если все зависимости по ресурсам перевести в его корень. Тогда получится дерево, как на рисунке 4.5.

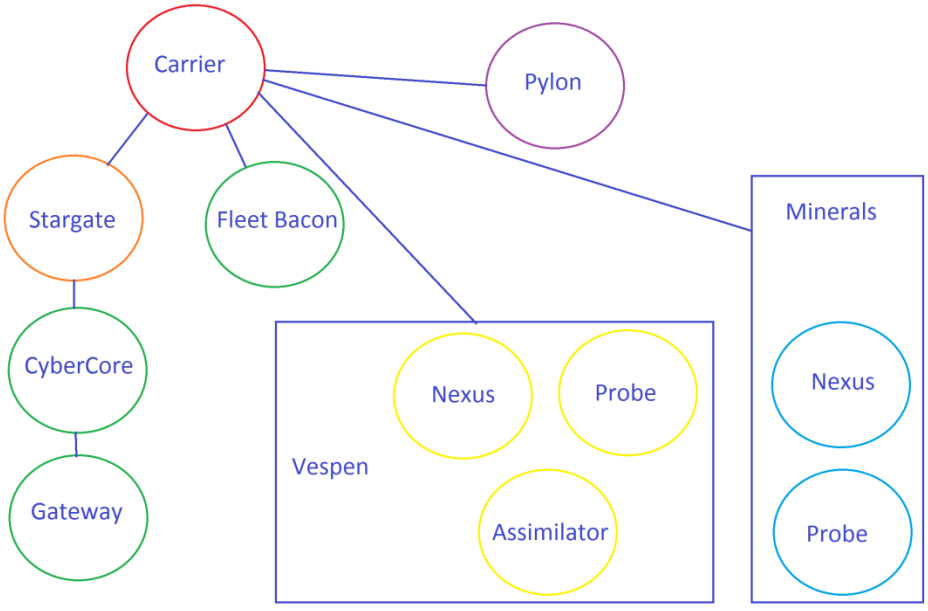

Рисунок 4.5 – Дерево зависимостей для авианосца после переноса всех ресурсных зависимостей в корень.

Поэтому, при получении награды, каждый боевой юнит смотрит на момент своего создания, и для всех его зависимостей, которые были живи в тот момент, частично пробрасывает награду вниз по дереву.

Но для того, чтобы это работало, необходимы еще 3 эвристики.

Первая — никакой юнит не может получить награды больше чем своя стоимость. Т.е. постройка юнита либо была выгодна в данный момент, либо нет. Если остается неизрасходованная награда, то она дается в момент получения за грамотный менеджмент.

Вторая — не боевые юниты, кроме технологических, одного типа получают награду по очереди, с большим приоритетом у тех, кто были построены первыми. Делается для того, чтобы не было перепроизводства рабочих и производящих зданий. У технологических зданий награду получает только первая живая постройка. Если ее награда полна — то значит, что награда остается в моменте ее получения, а не переходит следующему технологическому зданию этого типа. Делается это для той же самой причины.

Третья — чтобы агент пытался закончить игру, пусть в каждый момент времени будет отниматься награда, чтобы агент пытался быстрее закончить партию. Это очень распространенный трюк.

В этом случае статистика побед значительно улучшилась, что показано на рисунке 4.6.

AI	lose	draw	win
very_easy	0	3	7
easy	3	4	3
medium	7	2	1
medium_hard	9	1	0

Рисунок 4.6 – Статистика поражений, ничей и побед с распределением награды по дереву зависимостей.

4.4 Обобщенная оценка преимущества

Преимущество можно определить как меру того, в насколько хорошее состояние мы перейдем из текущего состояния, предприняв определенное действие. Для этого используются награды, которые были получены на каждом шаге. С помощью этих наград можно посчитать, какое преимущество было получено при выполнении выбранного действия. Таким образом, даже действие, которое само не принесло награду, но перевело состояние в выгодную позицию, получит положительную награду.

Чтобы рассчитать это, мы будем использовать алгоритм, известный как Обобщенная оценка преимущества, по следующим формулам[6].

Сначала вектор обобщенной оценки преимущества GAE инициализируется нулями.

$$GAE = 0$$

После чего считаются разности между оценкой текущего состояния, и его целью.

$$\delta = r_t + \gamma V(S_{t+1}) m_t - V(S_t)$$

Далее итеративно, начиная с последнего состояния в траектории, высчитывается вектор обобщенной оценки преимущества.

$$GAE_t = \delta + \gamma \lambda m_t GAE_{t+1}$$

Таким образом, считаются преимущества.

$$A_t = GAE_t$$

Чтобы получить цели оценок состояний, нужно воспользоваться следующей формулой.

$$\hat{V}(S_t) = GAE_t + V(S_t)$$

ГЛАВА 5 АЛГОРИТМЫ ОСНОВАННЫЕ НА ПОЛИТИКЕ

Цель агента при обучении с подкреплением - максимизировать ожидаемое вознаграждение при следовании политике π. Как и любая задача машинного обучения, мы определяем набор параметров θ (например, веса нейронной сети), чтобы параметризовать эту политику – π_θ.

Среди всех конечных Марковских процессов принятия решения существует хотя бы одна политика и, среди всех оптимальных политик, хотя бы одна стационарная и детерминированная. Но в нашем случае это скрытый Марковских процессов принятия решения.

Как и в других задачах машинного обучения, в случае, если мы сможем найти параметры модели, которые максимизируют МО награды, то мы решим задачу. Обычный подход к решению данной задачи – использование градиентного спуска.

Для этого распишем МО, и произведем внутри интеграла log derivative trick[7].

$$\nabla E_\pi[r(\tau)] = \nabla \int \pi(\tau) r(\tau) d\tau = \int \nabla \pi(\tau) r(\tau) d\tau \int \pi(r) \nabla log \pi(\tau) r(\tau) d\tau \ \#(5.1)$$

$$\nabla E_\pi[r(\tau)] = E_\pi[r(\tau) \nabla log \pi(\tau)] \ \#(5.2)$$

5.1 Алгоритм подкрепления

Прямая максимизация предыдущей формулы по Monte Carlo называется алгоритмом Reinforce.

Алгоритм подкрепления (также известный, как Монте-Карло оценка градиента политики) относится к классу алгоритмов обучения с подкреплением основанных на политике. Суть этого алгоритма в том, чтобы изменить логарифм правдоподобия в зависимости от кумулятивных наград, для того, чтобы ее максимизировать. Формула функции для максимизации:

$$\nabla E_{\pi_\theta}[r(\tau)] = E_{\pi_\theta}\left[\sum_{t=1}^{T} G_t \nabla \log \pi_\theta(a_t|s_t)\right] \#(5.3)$$

Алгоритм подкрепления быстро начинает обучаться, но из-за большой дисперсии обучение быстро проваливается, что показано на рисунке 5.1.

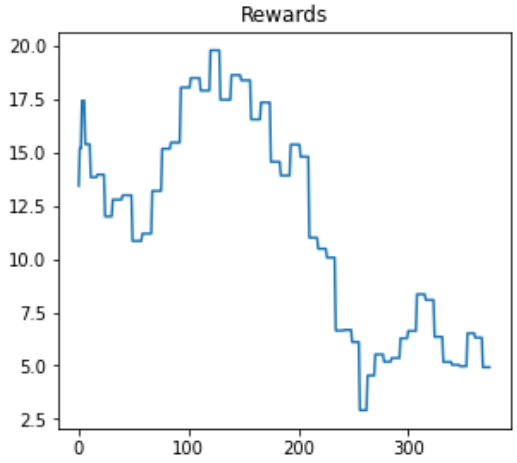

Рисунок 5.1 – График награды алгоритма подкрепления во время обучения.

Для того, чтобы понять, почему обучение так быстро проваливается, можно взглянуть на график энтропии на рисунке 5.2. Видно, что для многих состояний энтропися очень быстро падает. За один шаг агент может слишком сильно поменяться, и поэтому мы не можем гарантировать, что следующий агент будет лучше предыдущего. Более того, сильно страдает эксплоринг.

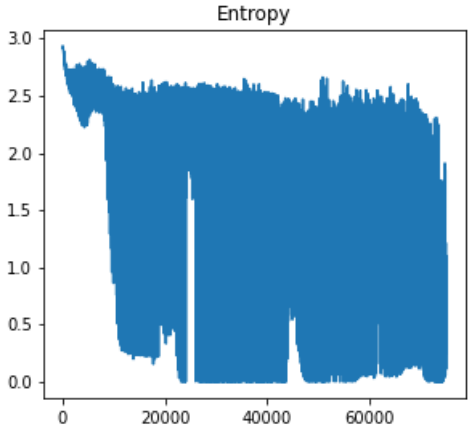

Рисунок 5.2 – График энтропии алгоритма подкрепления во время обучения.

Статистика действий на последней итерации этого короткого обучения расположена на рисунке 5.3.

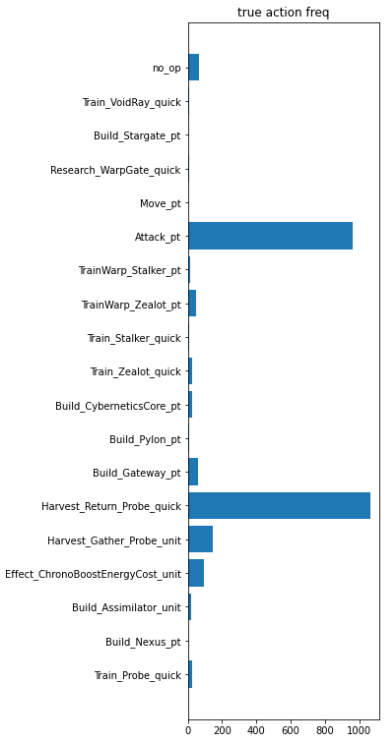

Рисунок 5.3 – Статистика действий алгоритма подкрепления на последней итерации.

5.2 Актор-критик с преимуществами

Актор-критик с преимуществами, как и алгоритм подкрепления, относится к классу алгоритмов обучения с подкреплением основанных на политике. У алгоритма подкрепления есть следующие проблемы: появляется большая дисперсия в пространстве логарифмов правдоподобия и кумулятивных наградах, что сильно портит градиенты и приводит к нестабильности в обучении. Актор-критик с преимуществами решает эту проблему за счет добавления в модель критика, который пытается оценивать состояние[8]. Градиент МО награды по политике и функция потери критики вычисляется по следующим формулам.

$$\nabla E_{\pi_\theta}[r(\tau)] = E_{\pi_\theta}[\sum_{t=1}^{T}(R_{t+1} + \gamma \mathrm{V}^{\omega}(\mathrm{S_{t+1}}) - \mathrm{V}^{\omega}(\mathrm{S_t}))\nabla \log \pi_\theta(\mathrm{a_t}|\mathrm{s_t})]\#(5.4)$$

$$CriticLoss = \left((R_{t+1} + \gamma \mathrm{V}^{\omega}(\mathrm{S_{t+1}}) - \mathrm{V}^{\omega}(\mathrm{S_t})\right)^2\#(5.5)$$

Если судить по графику награды, который расположен на рисунке 5.4, то результат получился примерно тем же самым, что и у алгоритма подкрепления.

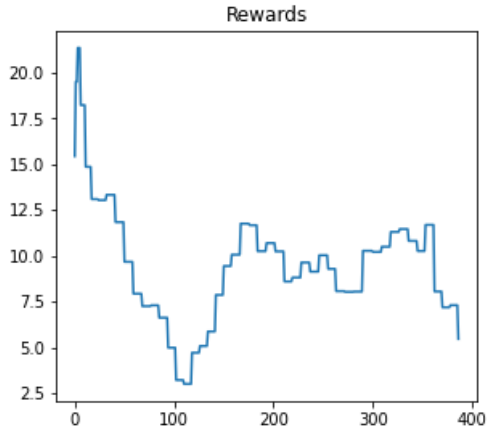

Рисунок 5.4 – График награды алгоритма A2C во время обучения.

То же самое можно проследить и по энтропии, очень быстро у многих состояний идет быстрый спад, что изображено на рисунке 5.5.

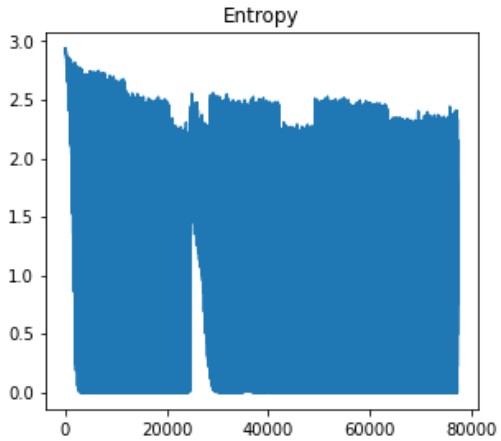

Рисунок 5.5 – График энтропии алгоритма Актор-критик с преимуществами во время обучения.

Понять, почему простое добавление критика не помогло, можно посмотрев на график потерь критика, который расположен на рисунке 5.6. Агент слишком сильно изменяется между итерациями, из-за чего он оказывался в очень различающихся состояниях на разных итерациях, из-за чего опыт критика с предыдущих итераций не очень полезен. Поэтому поведение агента схоже на поведение агента в алгоритме Reinforce.

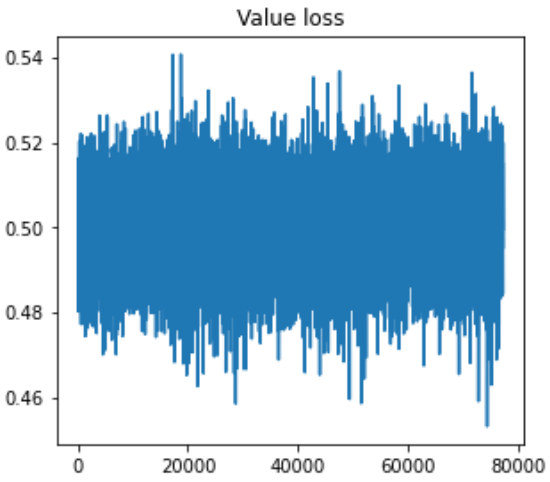

Рисунок 5.6 – График функции потери критика во время обучения алгоритмом Актор-критик с преимуществами.

Статистика действий на последней итерации алгоритма Актор-критик с преимуществами расположена на рисунке 5.7.

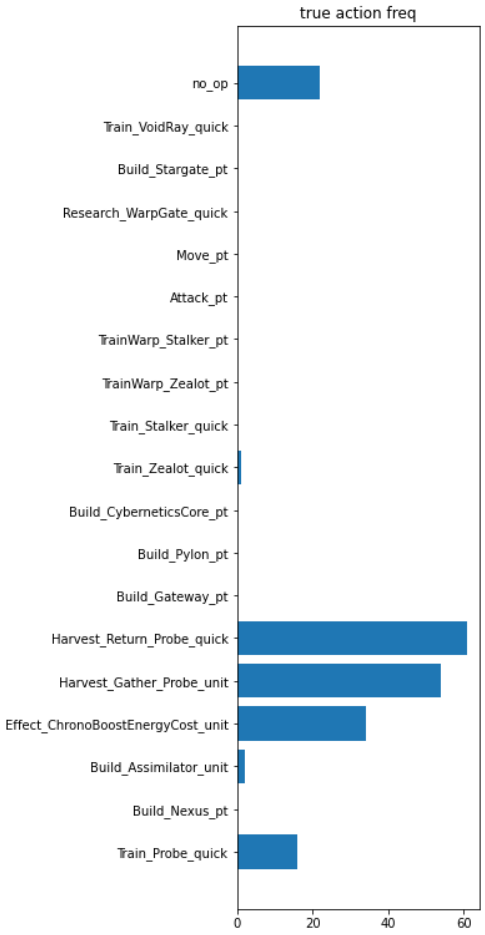

Рисунок 5.7 – Статистика действий алгоритма Актор-критик с преимуществами на последней итерации.

5.3 Проксимальная оптимизация политики

Проксимальная оптимизация политики - очередная веха в алгоритмах, основанных на политике. У методов основанных на политике есть проблема со сходимостью, которая произрастает из естественных свойств градиента. На практике градиентный спуск второго порядка включает в себя матрицу производных соответственно второго порядка, что делает этот алгоритм плохо масштабируемым для алгоритмов с большим количеством переменных, например нейронных сетей. Вычислительная сложность слишком высока для таких задач, из-за чего необходимы альтернативы. Хоть и проводятся многочисленные исследования по снижению сложности вычисления градиентов второго порядка, пока популярный способ это использование проксимальной оптимизации политики, который имеет несколько другой подход. Вместо наложения жесткого ограничения он формализует ограничение как дополнительное слагаемое в целевой функции либо делая ограничения в целевой функции. Мы можем в этом случае использовать оптимизатор первого порядка, т.е. градиентный спуск, для того, чтобы максимизировать целевую функцию. Даже с учетом неточности в приближении градиентов второго порядка, ущерб намного меньше, а вычисления получаются намного проще.

Краеугольный камень проксимальной оптимизации политики – доверительные регионы[9]. Т.е. за одну итерацию алгоритма проксимальной оптимизации политики мы пытаемся гарантировать, что политика останется внутри региона. После чего на следующей итерации поиск происходит в новом регионе.

В отличие от предыдущих алгоритмов, в проксимальной оптимизации политики градиенты будут пробрасываться через правдоподобие, а не логарифм правдоподобия.

$$ratio = \pi_{new}/\pi_{old} \ \#(5.6)$$

Отношение правдоподобия во время обучения к правдоподобию в момент разыгрывания траектории необходимо для того, чтобы не давать политике выбраться из доверительного региона.

$$p_1 = ratio * advantage$$
$$p_2 = clip(ratio, 1-\epsilon, 1+\epsilon) * advantage$$

$$ActorLoss = -min(p_1, p_2) \#(5.7)$$

Так же будем стараться, чтобы не было взрывных изменений у критика между итерациями. Для этого будем контролировать разницу между оценками состояний во время обучения и во время получения траектории.

$$L_1 = \left(\hat{V} - V_\theta(s)\right)^2$$

$$L_2 = \left(\hat{V}(s) - \left(V_{\theta_{old}}(s) + clip(V_\theta(s) - V_\theta, -\epsilon, \epsilon)\right)\right)^2$$

$$CriticLoss = \max(L_1, L_2) \#(5.8)$$

Итоговая функция потерь складывается из функции потерь критика, функции потерь политики и энтропии с отрицательным и небольшим по модулю отрицательным коэффициентом.

$$TotalLoss = CriticLoss + ActorLoss - \beta * Entropy \#(5.9)$$

В этом случае результат оказался намного лучше, что показано на рисунке 5.8.

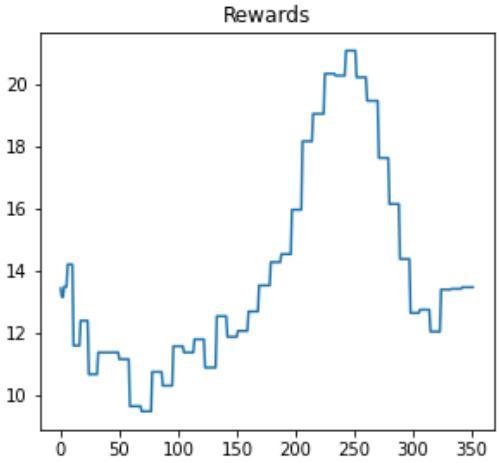

Рисунок 5.8 – График награды алгоритма проксимальной политики оптимизации во время обучения.

Причина этому то, что шаги делаются поэтапно, энтропия быстро не падает, что показано на рисунке 5.9, агент проходит по похожим состояниям.

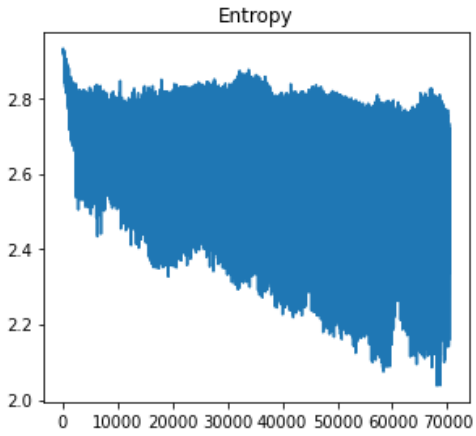

Рисунок 5.9 – График энтропии алгоритма проксимальной политики оптимизации во время обучения.

В том числе сходится и критик, что показано на рисунке 5.10.

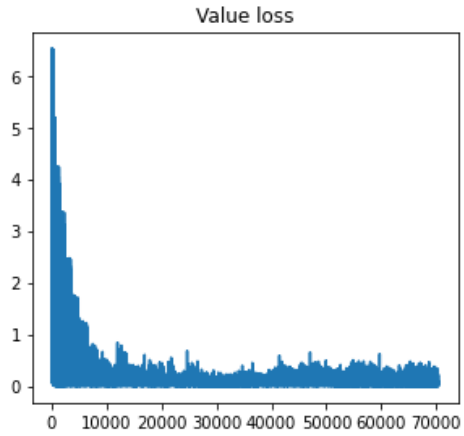

Рисунок 5.10 – График функции потери критика алгоритма проксимальной политики оптимизации во время обучения.

То, что эксплоринг сохраняется лучше можно просмотреть и по статистике действий, что показано на рисунке 5.11.

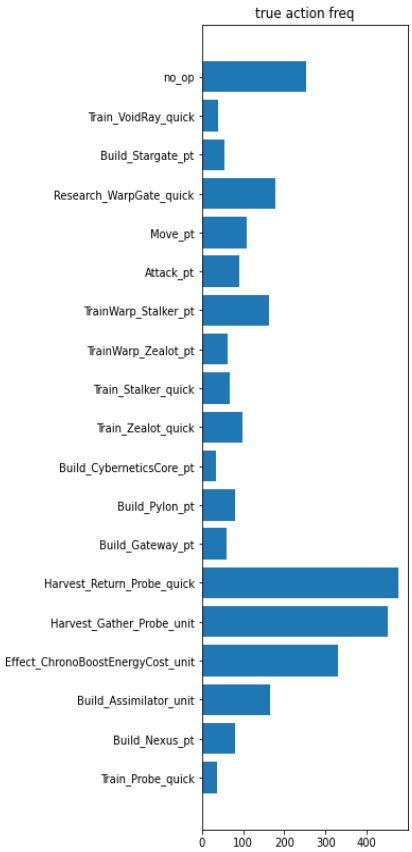

Рисунок 5.11 – Статистика действий алгоритма проксимальной политики оптимизации на последней итерации.

5.4 Проксимальная политика оптимизации с советчиком

Можно попытаться ускорить сходимость, введя своего рода советчика. В случае, когда функция потерь будет считать действие неудачным, то, вместо того, чтобы сразу распределять вероятности по всем действиям, можно спросить совета у советчика. Политика советчика должна отличаться от политики Actor-a, поэтому обучать его будем с небольшим шумом.

Возможен ли вариант того, что в случае, когда у актора и советчик неудачные действия, то научиться правильному действию нельзя? Это не так, т.к. совет от советчика к актору идет с небольшим коэффициентом, и, кроме того, советчик на следующей итерации поймет, что действие было неудачное, и уже не будет его советовать.

Как и в проксимальной политике оптимизации, необходимо использовать отношение правдоподобий, но функция потерь политики отличается.

$$ratio = \pi_{new}/\pi_{old} \ \#(5.10)$$

$$p_1 = ratio * advantage$$
$$p_2 = clip(ratio, 1-\epsilon, 1+\epsilon) * advantage$$
$$ActorLoss = -min(p_1, p_2) -$$
$$-mask * klCoef * \big(advantage * (advantage < 0)\big)$$
$$* KL(\pi_{new} | \pi_{adviser_{old}}) \ \#(5.11)$$

В то же время критик остается тем же, так как для советчика будет использоваться тот же критик, что и для политики.

$$L_1 = \left(\hat{V} - V_\theta(s)\right)^2$$
$$L_2 = \left(\hat{V}(s) - \left(V_{\theta_{old}}(s) + clip(V_\theta(s) - V_\theta, -\epsilon, \epsilon)\right)\right)^2$$
$$CriticLoss = \max(L_1, L_2) \ \#(5.12)$$

Основное отличие от политики заключается в функции потерь.

$$ratio_{adviser} = \pi_{adviser_{new}} / \pi_{adviser_{old}} \ \#(5.13)$$
$$p_{adviser_1} = ratio_{adviser} * advantage$$
$$p_{adviser_2} = clip(ratio_{adviser}, 1-\epsilon, 1+\epsilon) * advantage$$
$$AdviserLoss = (0.5 + 0.5U[0,1]) * min\left(p_{adviser_1}, p_{adviser_2}\right) \ \#(5.14)$$

Как и в PPO, необходимо использовать отношение правдоподобий, но функция потерь политики отличается.

$$TotalLoss = CriticLoss + ActorLoss + AdviserLoss - \beta * Entropy \ \#(5.15)$$

Эксперименты проводились, в том числе в другой среде, а именно Муджоко (Mujoco). В ней советчик помог улучшить результат проксимальной политики оптимизации, но для этого потребовалась подбирать коэффициент при дивергенции.

В том числе жизнеспособность этой модели можно проследить и по функции награды, которая изображена на рисунке 5.12

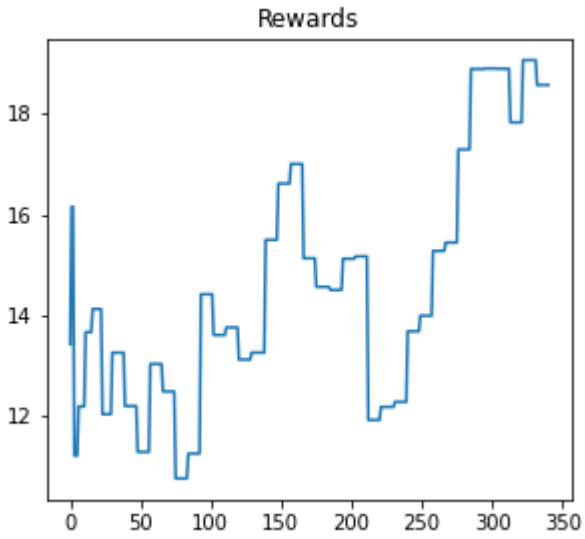

Рисунок 5.12 – График награды алгоритма проксимальной политики оптимизации с советчиком во время обучения.

Поведение графика энтропии, расположенного на рисунке 5.13, и графика функции потерь, расположенного на рисунке 5.14, критика схожи на поведение аналогичных графиков у PPO.

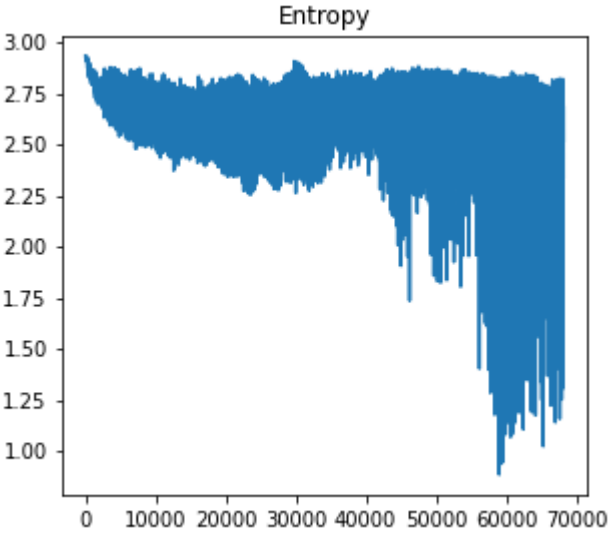

Рисунок 5.13 – График энтропии алгоритма проксимальной политики оптимизации с советчиком во время обучения.

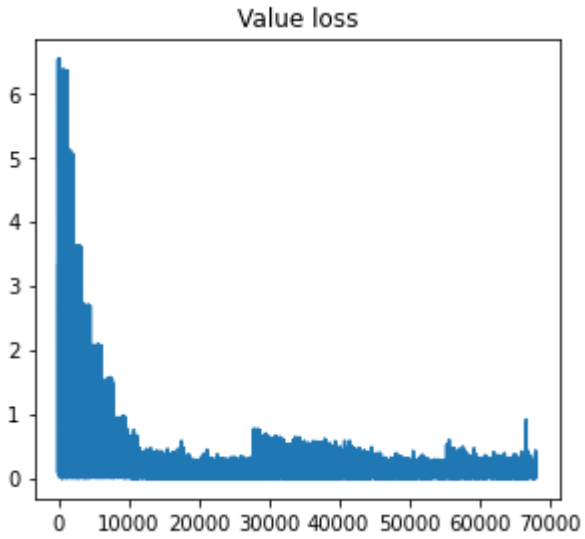

Рисунок 5.14 – График функции потерь критика алгоритма проксимальной политики оптимизации с советчиком во время обучения.

Статистика действий схожа на статистику действий у проксимальной политики оптимизации, что весьма логично. Политика быстро не выродилась в несколько типов действий. Это можно проследить по рисунке 5.15.

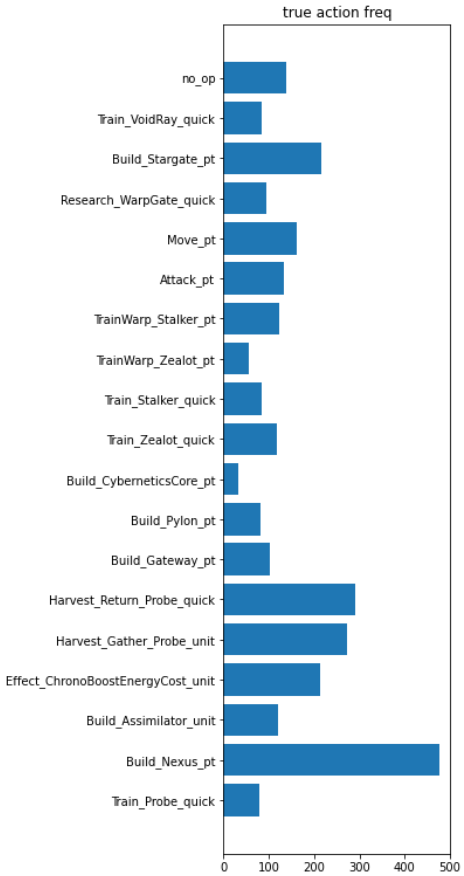

Рисунок 5.15 – Статистика действий проксимальной политики оптимизации с советчиком во время обучения.

5.5 Феодальная модель для иерархического обучения с подкреплением

Феодальные модели в обучении с подкреплением обладают следующими характеристиками: в них присутствует некоторая иерархия частей агента, причем прямая зависимость существует только между соседними уровнями. Каждое отношение зависимости включает в себя менеджера и рабочего. Менеджер общается с рабочим только постановкой цели, и награждает его только близостью состояния к поставленной цели, при этом менеджеру все равно на то, как рабочий будет достигать эту цель. Ему главное проверить то, чтобы через определенное время рабочий достиг цели.

Рабочий учится подчиняться, достигать цели, которые ему поставил менеджер. Менеджер, в свою очередь, ставит цели для рабочего в латентном пространстве.

Суть алгоритма: пусть цели, которые выдает менеджер имеют распределение фон Мизеса – Фишера[10]. С точностью до константы, в этом случае функцию потери можно расписать следующей формулой.

$$\nabla_\theta J(\pi_\theta) = E[\big(R_t - V(s_t)\big)\nabla d_{cos}(s_{t+c} - s_t, \mu(s_t)] \#(5.16)$$

Награда для менеджера используется напрямую из среды. Награда для рабочего имеет тем больше, чем поставленная цель ближе к разнице состояний в момент постановки цели и через некоторое количество шагов. Формула награды для рабочего расположена ниже.

$$r'_{t+c} = d_{cos}(s_{t+c} - s_t, g_t) \#(5.17)$$

Схема модели расположена на рисунке 5.16. Краткое описание выглядит следующим образом: наблюдение подается на вход в модуль восприятия, где из наблюдения формируется промежуточная репрезентация. Промежуточная репрезентация передается в менеджера и рабочего.

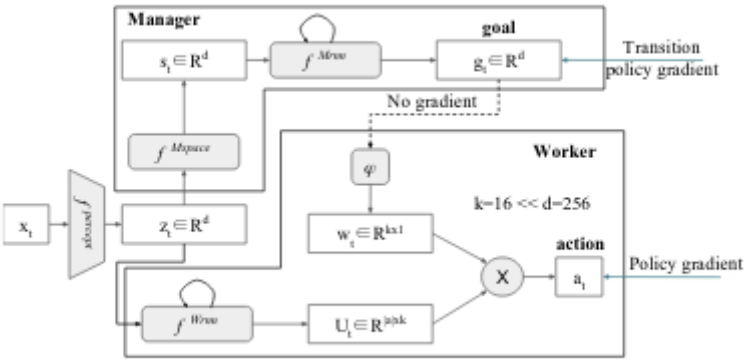

Рисунок 5.16 — Схема феодальной модели.

В менеджере промежуточная репрезентация проходит через модуль памяти, где она преобразуется в репрезентацию состояния. На основе репрезентации состояния формируется цель.

В рабочем промежуточная репрезентация преобразуется в матрицу размером в количество действий на размер латентного пространства. Некое количество предыдущих целей менеджера преобразуются в репрезентацию цели, после чего векторно умножается с полученной матрицей. Таким образом, получается распределение на действия. Кроме того, градиенты из рабочего не пробрасываются через репрезентации целей в менеджера.

С детерминированностью менеджера борются тем, что к репрезентации состояния добавляется шум. Это очень относительное решение проблемы, кроме того все еще остается проблема того, что невозможно сэмплировать из полученного распределения фон Мизеса – Фишера.

Ни в Муджоко, ни в Satrcraft II феодальная модель ни в одном эксперименте у меня не дала хоть сколько-то сравнимый результат с проксимальной оптимизацией политики. Только огромные изменения алгоритма смоги сделать так, что в Муджоко появился хотя бы более менее приемлемый результат, сравнимый с алгоритмом подкрепления. Но он все еще был сильно хуже, что проксимальной оптимизации политики, что проксимальной оптимизации политики с советчиком.

ГЛАВА 6 ПРОЦЕСС ОБУЧЕНИЯ

Рассмотрим процесс обучения для алгоритма проксимальной оптимизации политики с советчиком вместе с моделью, которая имеет усеченный набором действий без выбора параметров. Рассмотрен будет именно этот вариант, так как он оказался самым удачным.

Агент содержит в себе политику, которая выдает действия по текущему состоянию. Для обучения политика содержит в себе нейронную сеть, которую далее будем называть моделью. Для проксимальной политики оптимизации с советчиком модель делится на модель актора, модель советчика и модель критика. Модель актора и модель советчика выдают распределение по действиям, модель критика оценивает состояние. Более того, модель критика оценивает не только состояние по основной награде, но и по еще двум эксполринг наградам.

Раннер запускает среду и налаживает взаимодействие между агентом и средой. Во время взаимодействия агента и среды, раннер формирует траекторию. Траектория формируется из данных, которые необходимы для обучения, а именно: общая информация на каждом шаге, информация о состоянии карты, награда, эксполринг награда, улучшения, типы юнитов, принадлежность юнитов, выбранные действия, доступные действия, осуществленные действия, логарифмы правдоподобия выбранных действий, логарифмы правдоподобия осуществленных действий, логарифмы правдоподобия для советчика, оценки состояний награды и экспоринг награды. Общая информация на каждом шаге, информация о состоянии карты, улучшения и типы юнитов необходимы для описания текущего состояния. Награда и эксполринг награды нужны для получения преимуществ. Оценки состояний награды и экспоринг награды нужны для получения преимуществ и формирования функции потерь критика. Выбранные и осуществленные действия нужны для формирования функции потерь актора и советчика. Логарифмы правдоподобия действий во время обучения нужны для контроля того, чтобы между итерациями модель оставалась в доверительном регионе.

Для того, чтобы сформировать преимущества, награды, эксплоринг награды, оценки состояний, экспоринг оценки состояний передаются в модуль обобщенной оценки преимущества. Этот модуль формирует по этим данным преимущества и целевые оценки состояний по формулам, которые

расположены в главе про обобщенную оценку преимущества. После чего модуль обобщенной оценки преимущества дополняет этой информацией полученную траекторию.

Траекторию, которую формирует раннер, запрашивает сэмплер. Сэмплеру нужны траектории для формирования мини-батчей. Действует сэмплер следующим образом: при запросе данных для обучения, он поочередно выдает мини-батчи, повторно выдавая те же данные некоторое количество раз. Через некоторое количество итераций он просит раннер обновить траекторию. Раннер возвращает новую траекторию, и мини-батчи будут браться из нее.

Мини-батчи передаются в модуль, вычисляющий функцию потерь. Модуль функции потерь вычисляет соответственно функции потерь политики, советчика и критика.

Функция потерь политики принимает от модели актора текущий логарифм правдоподобия для выбранного и осуществленного действия, а из сохраненной траектории – логарифм правдоподобия во время обучения для выбранного и осуществленного действия. Итоговое преимущество считается как взвешенная сумма с некоторыми коэффициентами преимущества по обычной награде и двух преимуществ по экспоринг награде. Коэффициенты для взвешенной суммы меняются по ходу обучения.

Аналогично считается итоговое преимущество и функция потерь советчика. Отличие заключается только в том, что в функции потерь политики есть дивергенция между политикой и советчиком, в то время как у советчика присутствует некая доля случайности в преимуществах.

Функция потери критика состоит из трех частей, одна часть для обычной награды и две части для эксплоринг награды.

После того, как все функции потерь посчитаны, идет итерация градиентного спуска. Причем норму градиенты обязательно нужно контролировать.

Некоторая статистика из траекторий и модуля функции потерь логируются и отображаются в виде графиков. Для отображения были выбраны: функций наград, преимущества, оценки состояний, цели оценок состояний, функция потерь политики, функция потери критики, энтропии, нормы градиентов, статистика побед и поражений, дивергенция Кульбака-

Лейблера, статистика по действиям во время последней итерации. Пример графиков во время обучения изображен на рисунке 6.1.

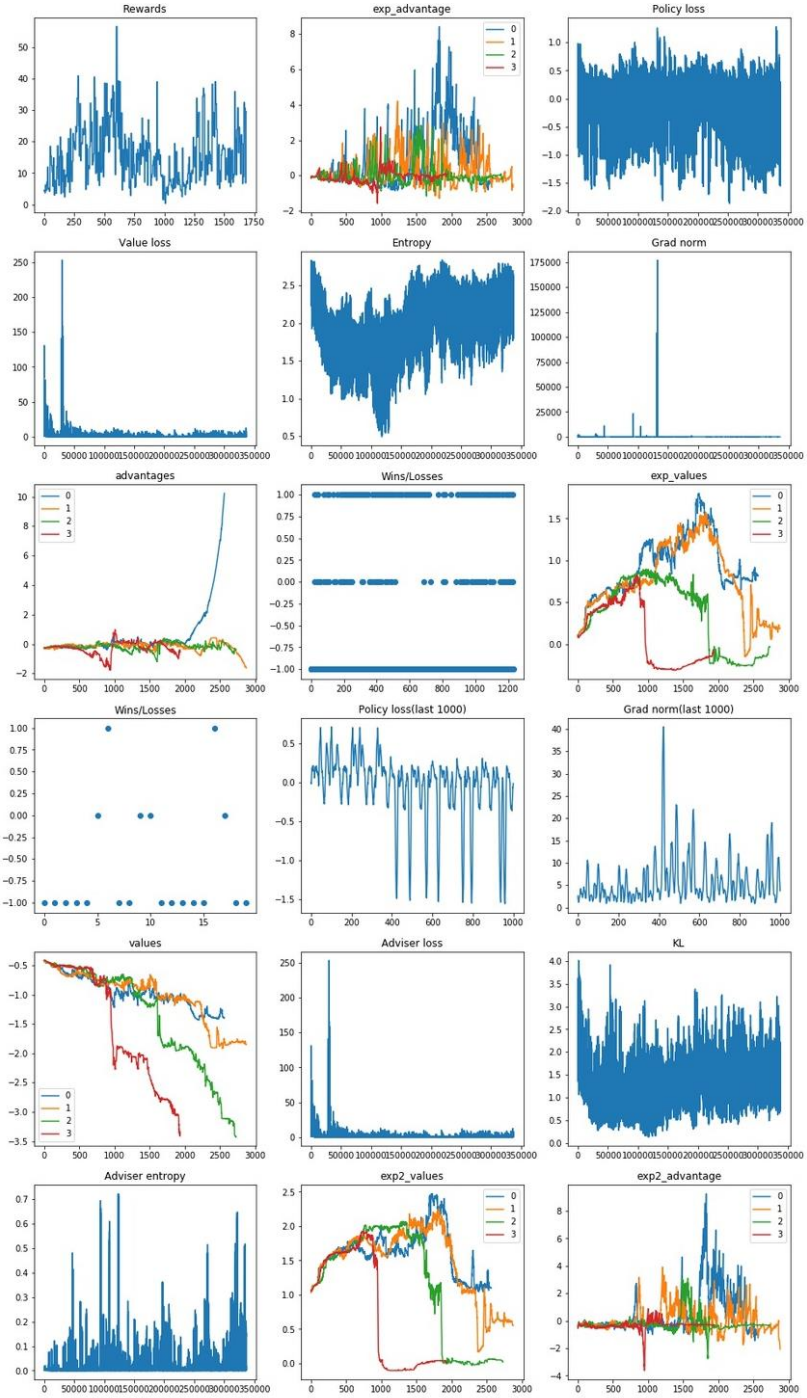

Рисунок 6.1 – Статистика во время обучения

Так же полезно знать о распределении действий, как показано на рисунке 6.2.

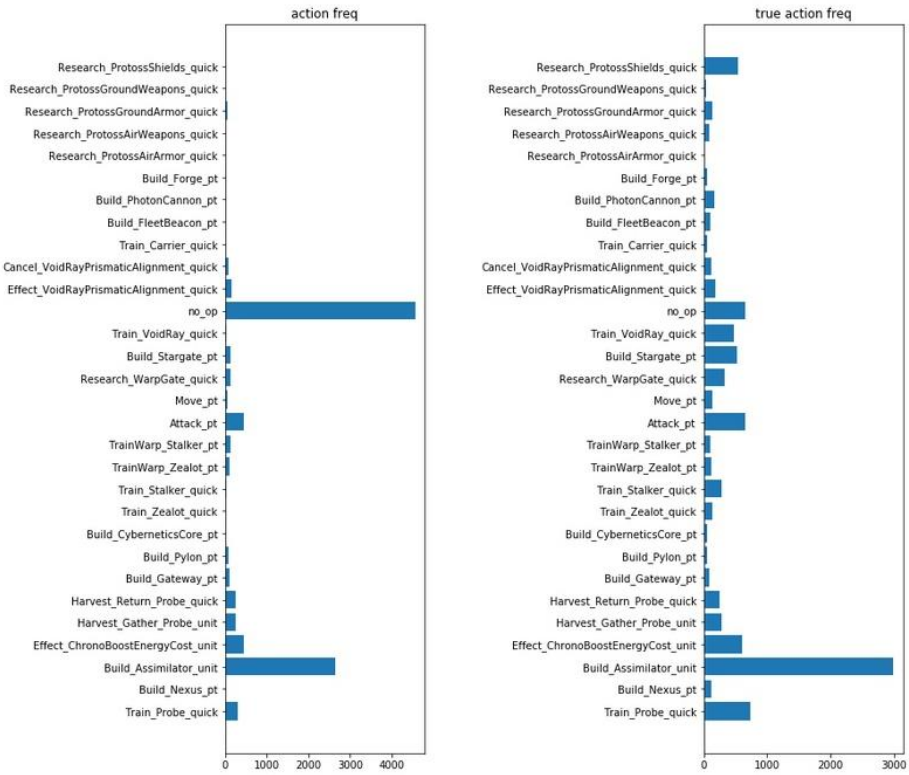

Рисунок 6.2 – Распределение действий на последней итерации.

Как было уже сказано, чтобы получить итоговое преимущество для политики и советчика нужно использовать взвешенную сумму. Первая экспоринг награда отвечает за экономику и производство без учета боевых действий, другая награда описана в главе о 4.3.Коэффициент для первой эксплоринг наград предполагается в начале обучения сделать достаточно большой, т.к. ее проще получать. По ходу обучения коэффициенты для первой эксплонг награды предполагается уменьшать, и через некоторое количество шагов, сделать его меньше, чем коэффициент для второй эксплоринг награды.

Траектория состоит из какого-то количества сыгранных партий. С учетом того, что память с предыдущих состояний передается через реккурентную нейронную сеть, то нужно следить за тем, чтобы память с предыдущих состояний всегда была правильной. Поэтому партия обучается целиком и последовательно, а при завершении прохода по партии память нужно сбросить и заново проинициализировать.

Обучение нейронной сети проходит на графическом процессоре, что сильно ускоряет процесс обучения.

В итоге статистика самого успешного агента изображена на рисунке 6.3.

AI	lose	draw	win
very_easy	0	1	9
easy	1	2	7
medium	4	3	3
medium_hard	8	0	2

Рисунок 6.3 – Статистика агента.

Даже в этой ситуации обучение в итоге может спотыкнуться по причине сложности среды и того, что максимизация побед это не то же самое, что максимизация взвешенной суммы основной награды с двумя экспоринг наградами. В этом можно убедиться по рисунку 6.4. А на рисунке 6.5 изображена победа агента против встроенного искусственного интеллекта средневысокой сложности.

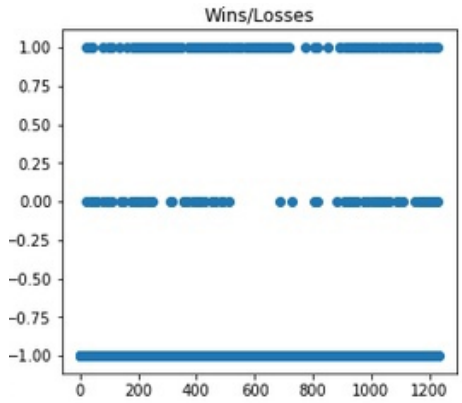

Рисунок 6.4 – Статистика побед, ничей и поражений агента.

Рисунок 6.5 – Победа агента над встроенным искусственным интеллектом средневысокой сложности.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы было изучено состояние обучения с подкреплением на текущий год, получен опыт проектировки алгоритмов обучения и опыт работы с библиотеками pysc2, pytorch.

Были реализованы алгоритмы подкрепления(Reinforce), актор-кртик с преимуществами (A3C, Advantage Actor Critic), проксимальная политика оптимизации (PPO, Proximal Policy Optimization), проксимальная оптимизация политики с советчиком (Proximal Policy Optimization with Adviser), феодальная модель для иерархического обучения с подкреплением (FeUdal Networks for Hierarchical Reinforcement Learning). В моделях с критиком использовалась обобщенная оценка преимущества (Generalized Advantage Estimation). Построена полная и две упрощенной модели. Упрощенные модель используется для ускоренного обучения за счет сокращения набора действий и уменьшения пространства наблюдений.

Удалось улучшить покрытие технологического дерева для необученного агента в среднем на 20-25% за счет того, что агент резервирует ресурсы и старается выполнять условия для недоступных действий.

За счет улучшенного эксплоринга среды, дополнительной спроектированной функции награды, оптимизации гиперпараметров, получилось обучить агента на упрощенной модели с помощью проксимальной оптимизации политики. Агент выигрывает очень легкий и легкий встроенный искусственный интеллект, но испытывает трудности со средним противником.

В ходе корректировки алгоритма и функции награды с учетом полученных оценок предыдущего агента, получилось улучшить результат за счет более умного распределения награды по дереву и добавления в модель советчика. Новый агент способен играть на равных со встроенным искусственным интеллектом средней сложности, но испытывает трудности с более сложными противниками. По сравнению с предыдущим ботом, его доля побед увеличилась в среднем на 15-20%.

Алгоритмы подкрепления и актор-критик с преимуществами не смогли показать хороший результат. Их обучение очень нестабильное и быстро вырождается в небольшой набор действий. Хоть они и быстро начинают обучаться, но итоговый результат оказался неудовлетворительным.

Еще хуже результаты у феодальной модели, которая смогла продемонстрировать слабую обучаемость даже при большом количестве доработок. Дополнительные эксперименты в среде Муджоко показали, что результаты данного алгоритма значительно хуже результатов проксимальной оптимизации политики.

Проксимальная оптимизация политики показала лучший результат. Добавление советчика при экспериментах в среде Муджоко помогло улучшить результат, поэтому советчик был включен в финальный алгоритм, где его добавление, после подбора гиперпараметров, помогло улучшить результат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Grandmaster level in StarCraft II using multi-agent reinforcement learning // Nature [Electronic resource]. – 2019. – Mode of access: https://www.nature.com/articles/s41586-019-1724-z. – Date of access: 12.10.2019.
2. The StarCraft Multi-Agent Challenge / Mikayel Samvelyan [etc] // arxiv [Electronic resource]. – 2019. – Mode of access: https://arxiv.org/pdf/1902.04043.pdf. – Date of access: 11.10.2020.
3. On Reinforcement Learning for Full-length Game of StarCraft / Zhen-Jia Pang [etc] // arxiv [Electronic resource]. – 2019. – Mode of access: https://arxiv.org/pdf/1809.09095.pdf. – Date of access: 17.11.2020.
4. Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces / Haotian Fu [etc] // arxiv [Electronic resource]. – 2019. – Mode of access: https://arxiv.org/pdf/1903.04959.pdf. – Date of access: 17.12.2019.
5. StarCraft II: A New Challenge for Reinforcement Learning / Oriol Vinyals [etc] // arxiv [Electronic resource]. – 2017. – Mode of access: https://arxiv.org/pdf/1708.04782.pdf. – Date of access: 10.02.2021.
6. High-Dimensional Continuous Control Using Generalized Advantage Estimation / John Schulma [etc] //arxiv [Electronic resource]. – 2015. – Mode of access: https://arxiv.org/pdf/1506.02438.pdf. – Date of access: 21.05.2020.
7. Policy Gradients in a Nutshell / Sanyam Kapoor // towardsdatascience [Electronic resource]. – 2018. – Mode of access: https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d. – Date of access: 23.07.2020.
8. Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU/ Mohammad Babaeizadeh [etc] //arxiv [Electronic resource]. – 2017. – Mode of access: https://arxiv.org/pdf/1611.06256.pdf. – Date of access: 16.10.2020.
9. Proximal Policy Optimization Algorithms / John Schulman [etc] //arxiv [Electronic resource]. – 2017. – Mode of access: https://arxiv.org/pdf/1707.06347.pdf. – Date of access: 19.01.2021.
10. FeUdal Networks for Hierarchical Reinforcement Learning / Alexander Sasha Vezhnevets [etc] //arxiv [Electronic resource]. – 2017. – Mode of access: https://arxiv.org/pdf/1703.01161.pdf. – Date of access: 20.02.2021.